

UNIVERSITY OF THE WITWATERSRAND

PHD THESIS

SCHOOL OF COMPUTER SCIENCE AND APPLIED MATHEMATICS

Play-style Identification and Player Modelling for Generating Tailored Advice in Video Games

Author

Branden Corwin Ingram

Supervisors

Prof. Benjamin Rosman

Prof. Clint Van Alten

Prof. Richard Klein

September 12, 2023



UNIVERSITY OF THE
WITWATERSRAND,
JOHANNESBURG

A thesis submitted to the Faculty of Science in fulfilment of the requirements of the degree of Doctor of Philosophy.

Abstract

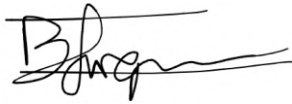
Recent advances in fields such as machine learning have enabled the development of systems that are able to achieve super-human performance on a number of domains, specifically in complex games such as Go and StarCraft. Based on these successes, it is reasonable to ask if these learned behaviours could be utilised to improve the performance of humans on the same tasks. However, the types of models used in these systems are typically not easily interpretable, and can not be directly used to improve the performance of a human. Additionally, humans tend to develop stylistic traits based on preference which aid in solving problems or competing at high levels. This thesis looks to address these difficulties by developing an end-to-end pipeline that can provide beneficial advice tailored to a player's style in a video game setting. Towards this end, we demonstrate the ability to firstly cluster variable-length multi-dimensional gameplay trajectories with respect to play-style in an unsupervised fashion. Secondly, we demonstrate the ability to learn to model an individual player's actions during gameplay. Thirdly we demonstrate the ability to learn policies representative of all the play-styles identified with an environment. Finally, we demonstrate how the utilisation of these components can generate advice which is tailored to the individual's style. This system would be particularly useful for improving tutorial systems that quickly become redundant lacking any personalisation. Additionally, this pipeline serves as a way for developers to garner insights on their player base which can be utilised for more informed decision-making on future feature releases and updates. For players, they gain a useful tool which can be utilised to learn how to play better as well identify as the characteristics of their gameplay as well as opponents. Furthermore, we contend that our approach has the potential to be employed in a broad range of learning domains.

Declaration

I declare that this thesis is my own, unaided work. It is being submitted for the Degree of Doctor of Philosophy at the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination at any other University.

Branden Corwin Ingram

BCI

A handwritten signature in black ink, appearing to read 'BCI', with a horizontal line drawn through it.

September 12, 2023

Acknowledgements

“Hey Look at us, Who woulda thought ... not me”¹. This quote very neatly sums up how I feel, having started university with average marks to now being where I am today. Although obvious, it should be stated how one does not simply go and complete a PhD. I consider myself fortunate to have encountered numerous remarkable individuals who have offered me their support, but none have been as significant as my trio of supervisors - Benji, Richard, and Clint. Over the years, all of you have evolved into much more than just supervisors or overseers. Instead, you have grown to become mentors, colleagues, and friends.

To Benji: It is strange I can not remember the first couple of times we met. I do, however, remember a lunch we had, where I proposed my idea for an AI in gaming interest group. Little did I know I was about to be drafted into this RoboCup project and how that would be the start of a truly amazing journey. Thanks to your guidance I have been able to see so many new places, meet so many amazing people, contribute to so much interesting research and personally grow as a researcher in my own right.

To Richard: No mere words can fully capture the debt of gratitude I owe you for your wise counsel and guidance over the years. I am deeply appreciative that you have been accessible and willing to converse at length about teaching, despite your many other obligations. It is undeniable that I would not have attained my current position without the opportunities that you generously provided.

To Clint: How far we have come from Honours, thank you for guiding me through my very first experience with research. Had you not made it so enjoyable and proposed a project that really suited me, I may have never gone on to Masters. I have always appreciated your consistent and reliable feedback across the eight years in total you have been supervising me. The speed at which these years have passed is remarkable, and I am fortunate to have had you as a supervisor throughout this entire journey.

Additionally, I would like to thank the members of CSAM and MSL for all their hard work, tireless efforts, and passion for the job, which made life as a student and now as a colleague that much easier. Specifically to Pravesh: It did not take too long sitting in your lessons all those years ago, to realise that yours would be my favourite course (Sorry Richard and Clint). However, what I did not know then was how wrong you are on, on so many miscellaneous topics, especially about Arsenal. Additionally, I did not know how annoyingly good you were at Squash. Thankfully among learning all these things I have learnt how similar we are and to share even the slightest trait with you can only be considered a good thing. To Steve: Somehow, Pravesh thinks we have a rivalry another one of his crazy ideas no doubt. All I said was that it irked me how you used my notes and advice after I wrote the Cloud Computing test to get 1% more than me and hence win a tablet. But joking aside thank you for running just a couple of years ahead with a reflective jacket so I had an easy target to aim for. I hope to see Kgomotso on campus more often because her smile brightens my day, even when I'm bothering her with random problems. The whole school owes her a debt of gratitude. To Ritesh, I am grateful for introducing me to the potential of academia as a viable career path, and I will always remember teaching BCO part-time with your guidance. Finally to MJ, although, it saddens me that you are no longer at Wits, I am indebted to how you would always go out of your way for me. I hope to see you on the basketball courts again one day for another game.

Although I was not a part of the RAIL Lab² since its inception, I have had the privilege of witnessing its remarkable growth into a machine learning powerhouse over the past four years. It has been an honour

¹<https://youtu.be/5QcIRo2eicQ>

²<https://www.raillab.org/>

to contribute to such an impressive group, which has also provided a pleasant work environment filled with delightful distractions. This is largely thanks to the other members, particularly Andries, Geraud, Tamlin, Mo, Manual, Jesse, Julian, and Nilesh. When you randomly receive a new, cool robot to play with, you know you're in the right place. To Michael, I believe everyone who knows you would agree that if they could match your work ethic even once a week, they would most likely be able to take a three-day weekend. I have greatly enjoyed our time working together on various projects, as well as our brief excursions onto the high-walls. I hope we can continue to work together in the future and that you develop a fondness for bouldering. To Dev, like Pravesh, I find that I can relate to you the most, whether it be from our shared experiences at an all-boys school or from other sources. The RAIL Lab would not have been the same without you, and board games with Josh would not have been nearly as enjoyable. In many ways, I see you as a better version of myself, and definitely a more convincing Elon lookalike.

I would like to take this opportunity to express my sincere gratitude to my friends for their unwavering support, encouragement, and friendship. Your humour, kindness, and empathy have made this journey more meaningful and enjoyable. I would like to extend special thanks to Leven and Gordon, who have been my confidants and advisors during challenging times. Your willingness to initiate contact and offer support has been invaluable and deeply appreciated. To Liron: If only I knew you knew I knew all along all along all along³ that you would become the greatest friend I would meet. I suppose I owe Steve for that one, had he not skipped out on that trip to China who knows what might have been. To Think I would have considered myself a happy person prior to our meeting, how naive I was to the joy and comedy that you would bring to my life. Oh and of course "The Boys" were it not for you, I would not have met so many people I now call dear friends. You are all positive role models in your own unique ways, and your unwavering support is a testament to your character. In particular, to Nikola: I may have taught you OS but you by far have contributed more to the betterment of myself, you are the true "moral carry ... no you". To Luka (A big man, the biggest of men no less⁴): Although I may have also taught you OS, you still found it heart to welcome me into your home like family. To continue the trend, I want to thank Ireton, who I have also taught OS, for always being enthusiastic to try new challenges with me whether physical or mental. I am greatly looking forward to all the milestones still to be accomplished with "The boys", now that "we" are all doctors.

In conclusion, I would like to express my gratitude to my family. Despite the challenges of the past five years, I feel blessed to have each of you in my life. Although we may be a small family, you embody the saying "quality over quantity". I extend my heartfelt appreciation to my parents for providing me with the opportunity, stability, support, and freedom to pursue my chosen path. Your unwavering patience with my occasional unwillingness to provide details on my life and your understandable worrying is not lost on me. I will undoubtedly exhibit the same level of concern with my own family someday. To my brother, I continue to look up to you, and even if you believe otherwise, that sentiment remains steadfast.

Of course, there have been many other people over the years who have all contributed in different ways, therefore to, "Et.al" I would like to offer my appreciation.

³<https://youtu.be/IHQr0HCIN2w?t=74>

⁴<https://youtu.be/ZgyU0LyWZ9M?t=172>

Contents

Abstract	i
Declaration	ii
Acknowledgements	iii
Table of Contents	v
Nomenclature	ix
1 Introduction	2
1.1 What are the different ways I can play a game?	3
1.2 What are the optimal ways to play in each style?	4
1.3 How do you as an individual play?	4
1.4 What should be advised?	5
1.5 Contributions	6
1.6 Overview	6
2 Background	8
2.1 Trajectory	8
2.2 Play-style	9
2.3 Neural Networks	11
2.4 Recurrent Neural Networks (RNN)	13
2.4.1 Long Short-Term Memory (LSTM)	15
2.4.2 Gated Recurrent Unit (GRU)	17
2.4.3 Bidirectional RNN	18
2.5 Autoencoders	19
2.5.1 Variational Autoencoders	20
2.5.2 Attention	21
2.6 Clustering	22
2.6.1 K-Means	23
2.6.2 GMM	24
2.6.3 Self-organising maps	25
2.7 Markov Decision Process	27
2.8 Reinforcement Learning	28
2.8.1 Q-learning	29
2.8.2 Deep Q Networks	30
2.9 Imitation Learning	31
2.9.1 Behavioural Cloning	32
3 Related Work	33
3.1 Giving Advice	33
3.2 Play-style Identification	36

3.2.1	Model-Based Approaches	37
3.2.2	Model-Free Approaches	38
3.2.3	Trajectory Clustering	38
3.3	Future State Prediction	40
3.3.1	Time-Series Forecasting	40
3.3.2	Modelling player actions	40
3.3.3	Opponent Modelling	41
3.4	Game Modelling	41
3.4.1	Reward-based	41
3.4.2	Data-based	42
3.5	Summary	43
4	Research Methodology	44
4.1	Play-style Identifier (PI)	45
4.2	Game Modeller (GM)	46
4.3	Player Modeller (PM)	47
4.4	Advisor (AD)	48
4.5	Domains	48
4.5.1	GridWorld	48
4.5.2	MiniDungeons	51
4.5.3	Mario	52
4.6	Summary	53
5	Play-style Identification	54
5.1	Introduction	54
5.2	Methodology	55
5.2.1	Trajectory Encoding	55
5.2.2	Trajectory Clustering	56
5.2.3	Cluster Analysis	57
5.2.3.1	Offline and Online Clustering Performance	57
5.2.3.2	Identifying Play-style Characteristics	59
5.2.3.3	Identifying Play-style Decision Boundaries	62
5.2.3.4	Identifying Mean Play-style Behaviours	64
5.2.4	Summary	65
5.3	Training	65
5.3.1	Autoencoder	66
5.3.2	Clustering	66
5.3.3	Baselines	66
5.4	Results	67
5.4.1	Offline Trajectory Clustering	67
5.4.2	Online Trajectory Clustering	68
5.4.3	Identifying Play-style Characteristics	71
5.4.3.1	Shared States	73
5.4.3.2	Differing States	75
5.4.3.3	Unique States	77
5.4.4	Identifying Play-style Decision Boundaries	79
5.4.5	Determining the Appropriate Number of Play-styles	82
5.4.6	Identifying Mean Play-style Trajectories	83
5.5	Conclusion	84

6	Game Modeller	85
6.1	Introduction	85
6.2	Methodology	86
6.2.1	Play-style-Centric Policy Generation (PCPG)	86
6.2.1.1	Trajectory Clustering (Step A)	87
6.2.1.2	Policy Learning (Steps B and C)	87
6.2.2	Training	87
6.2.3	Evaluation	88
6.3	Results	89
6.3.1	PCPG model performance	89
6.3.2	Behavioural and skill-based diversity	92
6.3.3	Clustering Performance	93
6.4	Conclusion	95
7	Player Modeller	96
7.1	Introduction	96
7.2	Methodology	97
7.2.1	Cluster Assisted Prediction (CAP)	97
7.2.2	Predictor Nodes	98
7.3	Offline Pre-Training	98
7.4	Online Fine-Tuning	99
7.5	Results	100
7.5.1	Offline Training Model Prediction Accuracy	100
7.5.2	Pre-training Effectiveness For Online Training	101
7.6	Conclusion	101
8	Advisor	102
8.1	Introduction	102
8.2	Methodology	103
8.2.1	Advice Generation	104
8.3	Experiments	106
8.4	Results	107
8.5	Conclusion	109
9	Conclusion	111
9.1	Limitations	112
9.1.1	Misclassification Rate	112
9.1.2	Continuous Action and State Space Domains	112
9.2	Future Work	113
9.2.1	Play-style Identifier (PI)	113
9.2.2	Game Modeller (GM)	113
9.2.3	Player Modeller (PM)	113
9.2.4	Advisor (AD)	114
9.3	Discussion	115
	References	116
A	Raw Data	133
A.1	GridWorld	133
A.1.1	E_1	133

A.1.2	E_2	134
A.1.3	E_3	135
A.1.4	E_4	136
A.1.5	E_5	137
A.2	MiniDungeons	137
A.3	Mario	138
B	Play-style Identification	140
B.1	Play-style Boundaries	140
B.1.1	E_1	140
B.1.2	E_2	141
B.1.3	E_3	142
B.1.4	E_4	143
B.1.5	E_5	144
B.2	Mean Behaviours	145
B.2.1	E_1	146
B.2.2	E_2	147
B.2.3	E_3	148
B.2.4	E_4	149
B.2.5	E_5	150

Nomenclature

Table 1: Symbols related to trajectories

Symbol	Description	Introduced
$\mathcal{D}$	Set of trajectories	Section 2.1
N	Number of data-points (trajectories) in a dataset	Section 2.6
X	Input Trajectory where $X_i \in \mathcal{D}$	Section 2.1
X'	Reconstructed input Trajectory	Section 2.5
P	Partial Trajectory $P = X[1 : t]$	Section 2.1
Z	Latent space representation of X	Section 2.5
$\mathcal{Z}$	The set of all latent space representations of $X \in \mathcal{D}$	Section 5.2
t	Timestep identifier $\in \mathbb{Z}$	Section 2.1
m	Length of an individual trajectory X	Section 2.1
x_t	Input state at a timestep t	Section 2.1
x'_t	Input state at a timestep t from a reconstructed trajectory X'	Section 5.2
x_m	Final state in a trajectory	Section 6.2
y	Ground truth label	Section 5.4
y', y_{pred}	Model predictions	Section 5.4
$s_{t,i}$	an individual timestep t within the i trajectory	Section 5.2

Table 2: Symbols related to clustering

Symbol	Description	Introduced
k	The number of clusters	Section 2.6
k_i	Cluster identifier k_i where $i \in 1, \dots, k$	Section 7.2
$\mathcal{P}$	Set of Partitions where $ \mathcal{P} = k$	Section 5.2
n	Number of data-points (trajectories) for a particular $\mathcal{P}_k$	Section 2.6
$\mathcal{C}$	Set of centroids for each partition	Section 5.2
$\mathcal{S}$	Finite set of states of all possible states	Section 2.7
f_k	State frequencies for all states in a particular cluster	Section 5.2
σ	A particular state	Section 5.2
v	decision boundary thresholding term	Section 5.2

Table 3: Symbols related to reinforcement learning

Symbol	Description	Introduced
$\mathcal{S}$	Finite set of states of all possible states	Section 2.7
A	Finite set of actions	Section 3.4
a_t	Action at a timestep t	Section 2.8
s_t	State at a timestep t	Section 2.8
r_t	Reward received for taking action a_t in state s_t	Section 2.8
P	Probability function	Section 2.7
R	Reward function	Section 2.7
γ	Discount factor	Section 2.7
π	Policy	Section 2.7
π^*	Optimal policy which maximises an expected cumulative reward	Section 2.7

Table 3: Symbols related to the Play-style Identifier, Game Modeller, Player Modeller and Advisor chapters

Symbol	Description	Introduced
$PI,$	The Play-style Identifier component	Section 4.1
$GM,$	The Game Modeller component	Section 4.2
$PM,$	The Player Modeller component	Section 4.3
$AD,$	The Advisor component	Section 4.4
π_k, GM_k	Play-style-centric policy	Section 6.2
π_k^*	Optimal play-style-centric policy	Section 6.2
p	Performance threshold	Section 6.2
$\mathcal{P}_{k,p}$	Further subset of $\mathcal{P}$ based upon p	Section 6.2
M_{ki}	Player model predictor node	Section 7.2
$\mathcal{U}_k$	Set of trajectories for a specific user	Section 7.2
$\bar{\mathcal{U}}$	Set of users	Section 8.2
ϖ	A random value in $[0, 1)$	Section 8.2
pk	Identified user's play-style	Section 8.2
a_e	Expert action from play-style-centric model	Section 8.2
a_e	player action from player model	Section 8.2
gm_{pk}	Play-style-centric policy associated with the identified user's style	Section 8.2

Chapter 1

Introduction

Learning new skills can often be a difficult, tedious and unintuitive process which results in people feeling disengaged and ultimately can lead to giving up. If we consider the case of someone playing a video game, a player might find a certain game or section of a game too difficult. This may be due to a number of reasons from lack of experience, inadequate tutorial or instructions, complex controls or even personal preferences. These factors could all lead to the player feeling disengaged and unsatisfied. One of the best ways for us as humans to combat this problem and better learn a skill is by receiving guidance from an expert. This is seen everywhere, from the school system to learning sports, or even as mentioned when playing video games. However, if we had a mechanism which could advise our players, not only could developers create more engaging video games, but from the player's perspective they would be able to improve ultimately leading to a sense of accomplishment. However, the challenge with advising is that there is no universal solution that works for everyone. Each person is unique having their own style of solving tasks, and what works for one person may not work for another [Bartle 1996]. Therefore formulating good advice requires developing an understanding of the individual such that we can tailor it to their specific needs and preferences. Referring back to our example of someone playing a video game if we as the advisor were to be able to identify their preferred style of play we could in turn offer advice appropriate to that case. In particular, if the player preferred to play defensively we could suggest strategies which help them become the optimal defensive player. However, while tailored advice can be valuable, providing it manually can be impractical due to time constraints and scalability issues. As a result, there is a growing trend towards using technology and automation to provide more efficient and effective advice.

Traditionally automatic advice generation has been handled through rule-based systems [Shortliffe *et al.* 1977] or decision tree-based systems [Bouza *et al.* 2008]. Both these techniques can be effective for generating automatic advice, but they have limitations. These systems have to be designed requiring domain knowledge and tend to be too general and inflexible, especially in complex environments. With regards to video games, automatic advice generation systems have traditionally been constrained to non-adaptive tutorial systems that employ a uniform approach across all users. This system quickly becomes redundant because they are often designed to be one-size-fits-all, and do not adapt to the player's preferences and increasing skill level [Robertson and Howells 2008]. We encapsulate these ideas of preferences, motivations, behaviour and skill level as a player's play-style. Therefore, to truly optimise the advice given to an individual, we need a system that is able to adapt to the changing play-style of a player.

The development of personalised automated advice requires a deep understanding of the user and their environment, which involves the complex task of data collection and analysis to determine their play-

style. More recently, machine learning techniques, such as neural networks and deep learning, have been used to tackle complex problems in video games [Mnih *et al.* 2015; OpenAI 2007; Jaderberg *et al.* 2019]. These techniques are well suited for analysing large amounts of data to find patterns because of their ability to automatically learn and adapt from input data, allowing them to identify complex relationships and patterns. For this reason, we look to apply such techniques to the problem of automatic tailored advice generation in video games. We look to approach this problem from the perspective of video games as they serve as good proxies for real-world learning environments as they often simulate complex scenarios, allowing players to engage in experiential learning and develop skills such as problem-solving [Robertson and Howells 2008]. Hence, creating a system that can offer customised guidance with positive outcomes could have effects in various fields, including education and learning in general.

The complexity of automatically generating customised advice in video games is evident, and we believe that it requires the implementation of various interconnected machine-learning techniques to overcome it. Thus, this is the problem that we aim to address in our thesis, through the formulation of four significant questions. Each of these is discussed below.

1.1 What are the different ways I can play a game?

There are often multiple ways to approach a game, and the choice of strategy can have a significant impact on the outcome. For example, in chess, there are many different openings, tactics, and endgame strategies that can be used to gain an advantage over the opponent. Therefore, identifying all possible play-styles in a game provides a more comprehensive understanding of the game mechanics, strategies, and tactics that can be used to play the game. Specifically, the benefits include:

- Enhanced game design: By identifying all possible play-styles, game developers can improve game design by creating more challenging and diverse game modes, rules, and challenges.
- Improved gameplay experiences: Understanding all possible play-styles can help players to develop their skills, learn new approaches, and challenge themselves, which can lead to a more immersive and enjoyable gaming experience.
- Prevention of cheating and exploitation: By having a comprehensive understanding of the different play-styles, game developers can better identify unusual or unfair behaviours that may negatively impact the game's integrity.

To answer this question, we may utilise techniques involving play-style identification used in video game analytics to gain an understanding and predict how individual players interact with a game. These techniques of play-style identification most commonly involved analysing player behaviour and preferences in order to cluster players into different categories based on their gameplay behaviour [Liu *et al.* 2013]. Using the previous example, a player may be clustered as aggressive if they tend to attack enemies head-on and take risks, while a defensive player may be more cautious and prefer to avoid direct confrontation. This clustering process, however, is not trivial as it requires the clustering of multi-dimensional, varying-length time-series data. Most approaches forgo processing this raw data and use only meta-data abstractions such as hours played, achievements unlocked or number of lives lost [Drachen *et al.* 2009]. We contend that by employing this approach, access to critical temporal information is lost, which is necessary for comprehending how play-styles evolve over time.

1.2 What are the optimal ways to play in each style?

The process of identifying an individual’s play-style within a video game is a necessary tool for generating tailored advice for that player. However, it is important to note that simply identifying a play-style alone is not sufficient to provide effective guidance. In particular, a critical aspect of generating effective advice is also knowing the optimal version of the player as it provides a target to reach for. Therefore creating a set of play-style-centric agents that play optimally according to a certain style would have several potential benefits, including:

- **Consistency:** By creating play-style-centric agents to play according to a certain style, you can ensure that they will always play in a consistent manner, which can be useful in scenarios where consistency is important, such as in training simulations or competitive gaming. Especially in the context of providing advice, inconsistency can lead to confusion or irritation of the user.
- **Diversity:** Creating a set of play-style-centric agents that play according to different styles can provide players with a more diverse and varied gaming experience, as they will be playing against opponents with different strengths and weaknesses.
- **Scalability:** Play-style-centric agents can be easily replicated and scaled up, which means that creating a set of play-style-centric agents that play according to a certain style can be a cost-effective way of providing a larger number of opponents for players to compete against.
- **Learning:** Play-style-centric agents that play according to a certain style can be useful for players who want to learn more about a particular strategy or play-style, as they can observe the agents in action and analyse their decision-making processes.

There are two common categories for determining solutions for complex tasks: supervised learning [Pearce and Zhu 2022] and reinforcement learning (RL) [Mnih *et al.* 2013]. These approaches typically focus solely on maximizing a score metric and do not consider the need to emulate a particular behaviour or play-style. While encoding play-styles into the scoring metric is a possible solution, it is often challenging and unintuitive [Arzate Cruz and Ramirez Uresti 2018]. Instead, we propose using data-centric approaches that learn from expert demonstrations. For example, by imitating the behaviours of aggressive and defensive-styled players, we can inform novices based on our prior understanding of their corresponding experts [Schaal 1999]. In practice, we look to imitate the styles identified through our play-style identification process. However, obtaining large amounts of data in video games necessary for deep learning techniques is not always feasible [Vinyals *et al.* 2019], and a data-efficient approach is required.

1.3 How do you as an individual play?

Understanding how an individual plays is crucial because it is essential to develop a model that can predict how the player will behave in a given situation to provide personalised advice. As a result, answering this question is a critical step towards tailoring the advice according to the individual’s playing style. Identifying a person’s play-style during live gameplay can, therefore, offer several benefits for both game developers and players. These benefits include:

- **Real-time Adaptation:** Online identification of a player’s play-style from partial trajectories allows game developers to adapt the gaming experience in real-time. Game developers can use this information to adjust the game mechanics and features on the fly, based on the player’s actions, preferences, and skill level, providing a more personalised and immersive gaming experience.

- **Improved Player Engagement:** Real-time identification of a player’s play-style from partial trajectories can lead to improved player engagement. By tailoring the gaming experience to the player’s play-style, game developers can create a more challenging, yet enjoyable experience, keeping players engaged and motivated to continue playing.
- **Dynamic Matchmaking:** Online identification of a player’s play-style from partial trajectories can also be used to create dynamic matchmaking. By matching players with similar play-styles, game developers can create more balanced and competitive matches, leading to a more enjoyable gaming experience for all players.

The problem of learning an understanding of an individual’s behaviour is commonly tackled in the field of player modelling [Drachen *et al.* 2009]. Player modelling is the process of creating a detailed representation of an individual player’s characteristics and behaviour based on their gameplay data. The major difficulty with using this type of model in an online fashion is that we require high accuracy with very little data. This is due to the inherent ambiguity present at the initial stages of temporal sequences, during which the distinct play-styles have not yet become discernible. This is especially important when generating advice as accurate predictions are needed to build trust and credibility between the advisor and the person seeking advice. If the advisor is able to accurately anticipate the person’s behaviour and provide helpful guidance, the person is more likely to see them as a valuable resource and seek their advice in the future. To combat this we look to employ pre-training techniques which are commonly used to enhance the performance of a model, especially when the model is complex, or the target dataset is small or different [Hendrycks *et al.* 2019].

1.4 What should be advised?

The ultimate objective of this thesis is to address the culmination of all the preceding elements, namely, how we can integrate these elements to develop personalised advice that users can leverage to enhance their performance. The benefits here are:

- **Enhanced player engagement:** Personalised advice can keep players engaged by providing feedback that is tailored to their specific needs, preferences, and skill level.
- **Improved player performance:** Tailored advice can help players improve their gameplay by highlighting areas for improvement and providing guidance on how to enhance their skills.
- **Increased player retention:** Providing personalised advice can keep players interested and motivated to continue playing, leading to increased retention rates.

Efficient and engaging advice-giving has been a long-standing challenge in various domains, such as teaching [Feiman-Nemser 1983], robotics [Kober *et al.* 2013], and video games [Shaffer *et al.* 2005]. Providing automatic *tailored* advice to individual users poses an even more challenging problem, particularly in video games. This requires generating advice that is not only helpful, relevant, and actionable for individual players but also overcomes obstacles such as personalisation and human-like reasoning [Brisson *et al.* 2012]. A promising approach to obtaining tailored advice and improving learning rates is to compare models of individual players with that of an expert, such as a teacher or mentor. Likewise, we aim to utilise both the player’s model and an expert’s model that aligns with their play-style. By analysing the disparities between the two models, our system can pinpoint areas where the novice player could improve and offer customised advice to assist them in improving their gameplay.

This work presents a novel approach for addressing the challenge of generating tailored advice for a user, as well as the absence of personalised tutorial systems in gaming. Our approach achieves several objectives in the context of video games. Firstly, it is capable of identifying all possible ways of playing

a game, which lays the foundation for further analysis. Secondly, it can identify an individual’s play-style, which is essential for generating tailored advice. Thirdly, it can learn the optimal ways of playing a game with respect to a particular play-style. Lastly, it can compare these models to compile the relevant information into tailored advice, which can be provided to the user. In summary, this work provides a comprehensive framework for generating personalised advice in video games, which can enhance the player’s overall experience. This results in improved performance for a user with respect to their identified style.

1.5 Contributions

Therefore, our main contribution comes in the form of an end-to-end multi-component system which learns from both a pre-existing dataset and an individual’s gameplay to compose tailored advice. This high-level design was presented at the doctoral consortium track at the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2021) [Ingram 2021]. Specifically, this overall model is made of the following contributions:

- **Play-style Identification:** In Chapter 5 we propose a novel method for identifying play-style in video games by clustering multi-dimensional variable-length trajectories, which represent different styles of gameplay behaviour. The model is evaluated on a benchmark dataset and a natural domain and can identify play-style from complete and partial (online) trajectories without requiring additional engineering or training time. Furthermore, we present novel techniques that employ state-based cluster analysis to uncover the traits and behaviours associated with individual play-styles, as well as their interrelationships. Portions of these ideas were presented at the IEEE Conference on Games 2022 (CoG 2022) [Ingram *et al.* 2022a] with further extensions published in the IEEE Transactions on Games (ToG) [Ingram *et al.* 2023b].
- **Game Modelling:** In Chapter 6, we introduce a novel application of Behavioural Cloning that involves the creation of play-style-centric policies. These policies are designed to exhibit specific behaviour, as identified in Chapter 5. Additionally, we demonstrate that our model can generate policies within each behavioural set with varying degrees of proficiency. This contribution was published in the Proceedings of the Eighteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2023) [Ingram *et al.* 2023a].
- **Player Modelling:** In Chapter 7, we introduce a unique multi-modular approach that enhances the precision of action prediction by utilising a recognised play-style. The proposed model incorporates a pre-training process that utilises an average pre-trained model, which corresponds to the identified play-style of the user. This feature is a distinctive advantage of our overall multi-component model. This contribution was presented at the IEEE Conference on Games 2022 (CoG 2022) [Ingram *et al.* 2022b].
- **Advice Generation:** In Chapter 8, we showcase the operability of the complete system in an online setting to produce tailored advice. This serves as an investigation into how these individual elements would be employed upon integration into a game. While not exhaustive, we demonstrate the proficiency of our methodology in reducing the divergence between a player’s policy and their expert counterpart.

1.6 Overview

The rest of the thesis is structured as follows, in Chapter 2 we survey the fundamental architectures and knowledge required for the understanding of our model. In Chapter 3 we present a comprehensive

overview of the existing literature and research related to our topic, including both prior work specifically focused on automatic advice generation as well as related work in neighbouring areas. Chapter 4 of this thesis contains a formal presentation of our overall model, including its functionality, as well as a formal introduction to the constituent components. This is followed by the four technical chapters (5, 6, 7 and 8) as discussed above. Finally, Chapter 9 summarises the key findings and contributions of the research and provides insights into their implications and potential future directions for research.

Chapter 2

Background

Broadly speaking, the task of creating tailored advice for gameplay necessitates drawing upon expertise from three core areas: temporal data analysis, clustering, and policy learning. For this research temporal information refers to time-series data which is represented by the sequence of actions an agent might perform, i.e. a trajectory. This chapter introduces the fundamental concepts and formalisms required for the understanding of the overall research.

Firstly, we introduce the fundamental concept of a trajectory in Section 2.1. This trajectory data serves as the basis for extracting valuable information to identify playstyles, as discussed in Section 2.2. Section 2.3 introduces the idea of neural networks which is utilised extensively by all components for the overall framework. Additionally, Section 2.4 provides an overview of various recurrent neural network architectures, while Section 2.5 introduces the concept of an autoencoder. These architectures are necessary for the processing of temporal data. Section 2.6 then describes the different approaches to unsupervised clustering, where data is separated based upon some underlying features into a fixed number of groups. Lastly, Sections 2.7, 2.8 and 2.9 outline in detail the learning methods required to generate our simulated data and learn policies required for creating play-style-centric agents.

2.1 Trajectory

In mathematics, a time series is a series of data points indexed (or listed or graphed) in time order. Most commonly, a time series is a sequence of data points taken at successive equally spaced points in time. Thus it is a sequence of discrete-time data. We define a trajectory X in Equation 2.1 such that:

$$X = (x_1, x_2, \dots, x_t, \dots, x_m), \quad (2.1)$$

where m is the number of time steps. Here each data point x_t is an element in trajectory X which represents the context or state of an environment at that particular point in time. In the upcoming section (Section 4.5), we will provide a detailed account of the unique attributes of the context in each of the assessed domains, and also describe the procedure of collating multiple trajectories to create our diverse datasets denoted by $\mathcal{D}$.

Furthermore, we define a partial trajectory P as a sub-sequence of states from a trajectory X starting with the first element (x_1) and including all elements up until the final element (x_t) where t is some time-step as defined in Equation 2.2. For the purpose of this thesis, we use the following notation ($P = X[1 : t]$) for referencing a partial trajectory from X such that:

$$P = (x_1, x_2, \dots, x_t). \quad (2.2)$$

In actuality, trajectories consist of a series of states across multiple dimensions. Nevertheless, Figure 2.1 portrays a set of trajectories in two dimensions arbitrarily chosen for visualisation purposes.

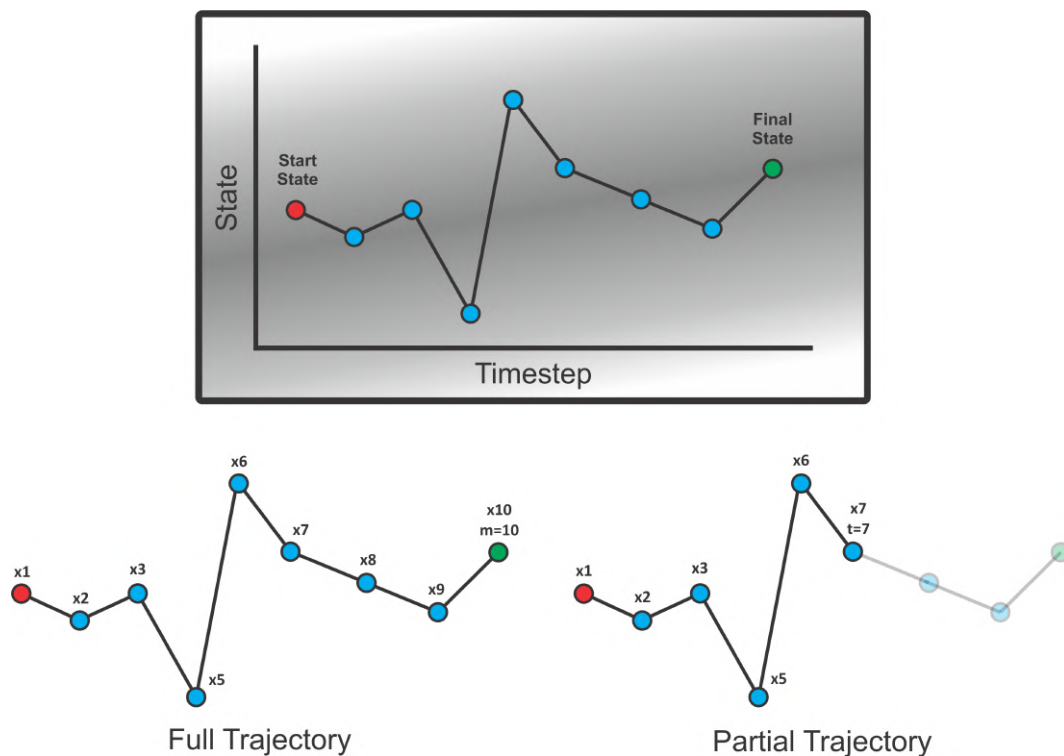


Figure 2.1: Visualisation of both a full and partial trajectory.

2.2 Play-style

A play-style refers to the characteristic way in which a player approaches a game or activity, with specific strategies and behaviours that are consistent and recognisable [Sirlin 2006]. A play-style can vary depending on the individual, the game or activity being played, and the context in which it is being played. The concept of play-styles has evolved over time and has been discussed by many different authors and researchers in the context of behavioural patterns and game strategy [Tzu 2020], [Tekinbas and Zimmerman 2003].

Formally describing a play-style involves identifying the key elements of the player's approach, such as their preferred tactics, level of aggression or caution, decision-making process, and overall attitude towards the game. This might also involve analysing the player's strengths and weaknesses, as well as any patterns or tendencies they exhibit in their play.

For example Bartle's [2004] taxonomy of player types was developed to describe different player motivations and play-styles in the context of online multiplayer games, specifically MUDs¹ (multi-user dungeons). The four player types identified by Bartle [2004] are:

- **Achievers:** Players who are motivated by achieving goals and progressing through the game's content. They are often highly competitive and seek to accumulate status, power, and rewards.

¹<https://medium.com/@williamson.f93/multi-user-dungeons-muds-what-are-they-and-how-to-play-af3ec0f29f4a>

- **Explorers:** Players who are motivated by discovering and learning about the game’s mechanics and content. They enjoy uncovering hidden areas, secrets, and easter eggs and are often motivated by the thrill of discovery.
- **Socialisers:** Players who are motivated by social interaction and building relationships with other players. They enjoy chatting, forming groups, and participating in in-game social events.
- **Killers:** Players who are motivated by conflict and competition with other players. They enjoy engaging in combat and PvP (player vs. player) activities, and are often motivated by the thrill of victory and the satisfaction of defeating opponents.

Each of these player types represents a distinct play-style, with its own set of motivations, goals, and behaviours. Players may exhibit characteristics of multiple player types, but tend to have a dominant type that reflects their primary motivation for playing the game. More recently, Grapevine [2020] associated play-styles with a player’s choice of character in the video game *Super Smash Bros. Ultimate*² as seen in Figure 2.2.



Figure 2.2: Play-style characterisation for playable characters in the game *Super Smash Bros. Ultimate* as defined by Grapevine [2020]

Here Grapevine [2020] categorised the play-styles in *Super Smash Bros. Ultimate* into several types:

- **Zone breakers** are characters with versatile abilities that allow them to pressure opponents while maintaining a safe distance.

²<https://www.smashbros.com/>

- **Mix-up** characters have a broad range of moves, but lack the safety of rushdown characters, allowing them to switch between bait and punish/zoning play-styles and aggressive ones.
- **Footsies** characters rely on their strong ground game and explosive power.
- **Hit and Run** characters have speed and a toolkit that lets them rush in, deal damage, and retreat before the opponent can counterattack.
- **Half-Grapplers** have move-sets that emphasise what they can get off of a grab, and Trappers have a heavy projectile game that limits the opponent's available space.
- **Turtles** are defensive characters with long-range tools designed to poke opponents from afar.
- **Dynamic** characters have unique abilities that define their play-style, such as Shulk's reliance on the active Monado or Pokemon Trainer's use of different Pokemon for various needs.

Ultimately, a formal description of a play-style would seek to capture the unique qualities that define a player's approach and provide insight into their motivations, strengths, and weaknesses as a player. We look to learn to identify such qualities in an unsupervised fashion for the ultimate purpose of advice generation.

2.3 Neural Networks

Artificial neural networks (NN) is a mathematical model that is designed to recognize patterns in data, by simulating the way the human brain works [McClelland *et al.* 1986]. It consists of a large number of interconnected processing nodes, called neurons, that are organized into layers. Each layer receives input data and performs mathematical operations to transform the data into a more meaningful representation. The output of one layer is then passed on as input to the next layer, and this process continues until the final output is produced. Figure 2.3 depicts a traditional fully connected NN with an input vector of 3 values, two hidden layers of four nodes each and a single output node.

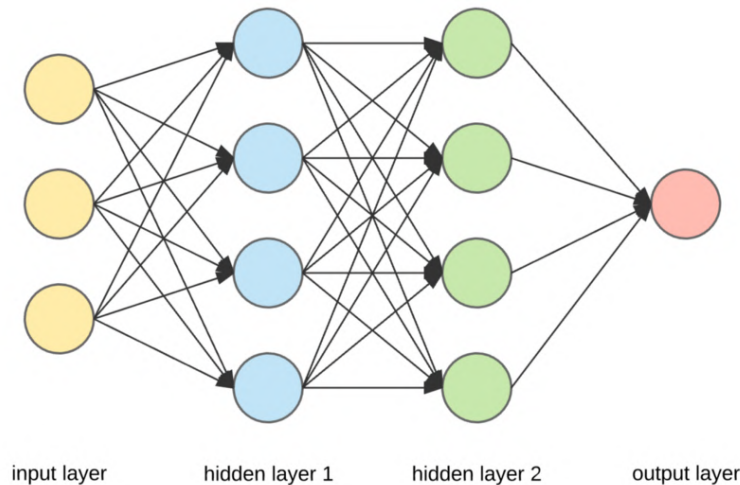


Figure 2.3: Example of simple NN

Neural networks are a fundamental concept in machine learning and artificial intelligence, and they form the basis for many advanced techniques including recurrent neural networks (RNNs) described in Section 2.4.

In a neural network, we are trying to learn a function that maps inputs to outputs. Mathematically, we can represent this function as $f(X)$, where X is the input and $f(X)$ is the output. To learn this mapping a NN undergoes an iterative training procedure. We illustrate this procedure by considering a simple fully connected feedforward neural network with one input layer, one hidden layer, and one output layer. The network is trained on a dataset with n input examples, each of which has d features.

The mathematical representation of the network is as follows [Svozil *et al.* 1997]:

- **Initialization:** The weights of the connections between the neurons are initialized randomly to small values, typically between -1 and 1.

$$\begin{aligned} W_1 &= \text{random}(d, h) \\ b_1 &= \text{zeros}(h) \\ W_2 &= \text{random}(h, 1) \\ b_2 &= \text{zeros}(1) \end{aligned} \tag{2.3}$$

where W_1 is the weight matrix connecting the input layer to the hidden layer, b_1 is the bias vector for the hidden layer, W_2 is the weight matrix connecting the hidden layer to the output layer, and b_2 is the bias for the output layer. Here d is the number of input features, h is the number of hidden units, and there is 1 output node.

- **Forward propagation:** The input data is fed into the input layer of the neural network, and the data is propagated forward through the network. Each neuron in each layer performs a calculation based on the input it receives and its associated weight. The output of each neuron is then passed on as input to the next layer, until the final output is produced.

$$\begin{aligned} z_1 &= X \cdot W_1 + b_1 && \text{neuron value at hidden layer} \\ a_1 &= \text{sigmoid}(z_1) && \text{activation value at hidden layer} \\ z_2 &= a_1 \cdot W_2 + b_2 && \text{neuron value at output layer} \\ y_{\text{pred}} &= \text{sigmoid}(z_2) && \text{activation value at hidden layer} \end{aligned} \tag{2.4}$$

where X is the input matrix of shape (n, d) , sigmoid is the activation function used for the hidden and output layers, and y_{pred} is the predicted output matrix of shape $(n, 1)$.

- **Calculate loss:** The difference between the predicted output and the actual output is calculated using a loss function, such as mean squared error (MSE) [Wallach and Goffinet 1989] or cross-entropy [Zhang and Sabuncu 2018].

$$\text{loss} = \text{MSE}(y_{\text{true}}, y_{\text{pred}}) \tag{2.5}$$

where y_{true} is the true output matrix of shape $(n, 1)$.

- **Backpropagation:** The error signal is propagated backwards through the network, and the weights of the connections are updated using an optimisation algorithm such as gradient descent [Ruder 2016]. This involves calculating the gradient of the loss function with respect to each weight in the network, and then adjusting the weights in the opposite direction of the gradient in order to

minimise the loss.

$$\begin{aligned}
\delta_2 &= \text{loss}(y_{\text{pred}}, y_{\text{true}}) \cdot \text{sigmoid_derivative}(z_2) \\
\delta_1 &= \delta_2 \cdot W_2^T \cdot \text{sigmoid_derivative}(z_1) \\
dW_2 &= a_1^T \cdot \delta_2 \\
db_2 &= \sum_{i=1}^n \delta_{2_i} \\
dW_1 &= X^T \cdot \delta_1 \\
db_1 &= \sum_{i=1}^n \delta_{1_i} \\
W_2 &= W_2 - \alpha \cdot dW_2 \\
b_2 &= b_2 - \alpha \cdot db_2 \\
W_1 &= W_1 - \alpha \cdot dW_1 \\
b_1 &= b_1 - \alpha \cdot db_1
\end{aligned} \tag{2.6}$$

where δ_2 is the error signal for the output layer, δ_1 is the error signal for the hidden layer, dW_2 is the gradient of the loss function with respect to the weights connecting the hidden layer to the output layer, db_2 is the gradient of the loss function with respect to the bias for the output layer, dW_1 is the gradient of the loss function with respect to the weights connecting the input layer to the hidden layer, and db_1 is the gradient of the loss function with respect to the bias for the hidden layer. The learning rate (α) is a hyperparameter that controls the step size of the weight updates.

- **Repeat:** Steps 2-4 are repeated for each input in the training set, until the network has been trained on all of the data. This is known as an epoch.

The performance of the network is evaluated on a separate set of data, called the testing set, in order to ensure that the network has not overfit the training data. Overall, the ability of neural networks to process large amounts of data, identify patterns, and learn from feedback makes them an important tool for automatic advice generation.

2.4 Recurrent Neural Networks (RNN)

A limitation of standard neural networks is that they require fixed-length input and are not able to interpret any temporal relationships. A recurrent neural network (RNN) [Rumelhart *et al.* 1986] is a type of neural network that is particularly well-suited for processing sequential data, such as time series, speech signals [Graves *et al.* 2013], and text [Tealab 2018]. The defining characteristic of an RNN is that it allows previous outputs to be used as inputs when generating the current output, giving the network a form of memory, which enables it to save the states or details of prior inputs. While traditional deep neural networks assume that inputs and outputs are independent of each other, the output of recurrent neural networks depends on the prior elements within the sequence. An RNN consists of a series of hidden units, where each hidden unit takes in the current input and the previous hidden state as inputs and produces an output, which is used as the input for the next hidden unit. Therefore, the hidden state is the output of a hidden unit, and it captures information about previous inputs that is stored in the memory of the RNN. This allows the network to maintain information about the previous states in the sequence, which can be useful in making predictions about the next state in the sequence.

To train an RNN, the network is unrolled over the length of the input sequence, as seen in Figure 2.4 which uses the following notation:

- X : is an entire input trajectory
- x_t : is the input at time step t ($x_t = X[t]$)
- U : are the weights associated with the input state of the recurrent layer
- h_t : is the state of the hidden vector (memory) at time step t
- W : are the weights associated with the output state of the recurrent layer
- O_t : is the output of the network at time step t
- V : are the weights associated with the hidden state of the recurrent layer

Unrolling refers to the process of expanding the network over time, by creating a series of connected copies of the same RNN cell, one for each time step of the input sequence. This process allows the network to be viewed as a NN, where each RNN cell represents a layer in the network and the weights can be updated using the standard backpropagation algorithm. During training, the loss is computed at each time step based on the difference between the predicted output and the true output, and the gradients are computed using backpropagation through time (BPTT) [Werbos 1990], which is an extension of backpropagation to recurrent neural networks. The weights are then updated using gradient descent or a variant thereof. However, the gradients can be difficult to compute in an RNN due to the repeated use of the same parameters in each hidden unit, which can result in gradients that either explode or vanish. This is known as the vanishing gradient problem [Hochreiter 1998].

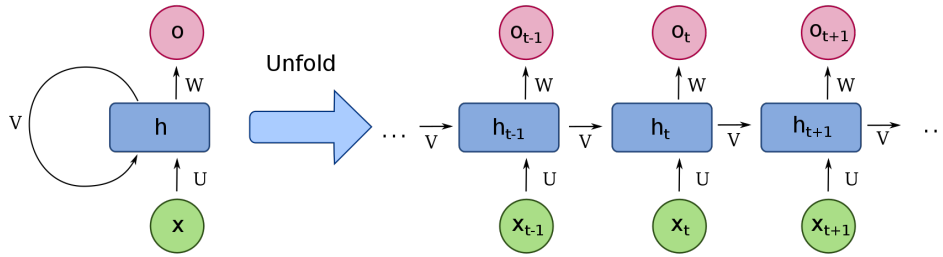


Figure 2.4: Standard RNN architecture; Compressed (left), Unrolled (right). Here X is an entire input trajectory, x_t is the input at time step t ($x_t = X[t]$), h_t is the hidden state vector and O_t is the output vector at time step t . Finally U and W are the weight matrices between the input and hidden vectors as well as between the hidden and output vectors respectively.

RNNs, like other neural networks, employ the mechanisms of forward and backward propagation to learn from data and make predictions. These mechanisms can be described as follows:

Forward propagation:

- At each time step t , the RNN takes in an input vector $x_t = X[t]$ and the previous hidden state h_{t-1} to produce a new hidden state h_t and an output O_t .
- The input vector x_t is multiplied by a weight matrix U that represents the connections between the input layer and the hidden layer. This product is added to the product of the previous hidden state h_{t-1} and a weight matrix V that represents the connections between the hidden layer at time $t - 1$ and the hidden layer at time t . The resulting sum is passed through an activation function to produce the new hidden state h_t .

- The hidden state h_t is then multiplied by a weight matrix W that represents the connections between the hidden layer and the output layer. This product is passed through another activation function to produce the output O_t .

Backward propagation:

- To train the RNN, we need to compute the gradients of the loss with respect to the weights U , W , and V .
- Starting from the final time step m , we compute the error gradient of the loss with respect to the output O_m . This gradient is then backpropagated to the previous hidden state h_{m-1} and the weights V using the chain rule of differentiation.
- The error gradient is then propagated backwards through time, by computing the error gradient at each time step t in terms of the error gradient at the next time step $t + 1$, and backpropagating it to the previous hidden state h_{t-1} and the weights U and W using the chain rule.
- Finally, the gradients are used to update the weights using an optimisation algorithm such as stochastic gradient descent (SGD).

There are several variants of RNNs that have been developed to address this issue, including long short-term memory (LSTM) networks and gated recurrent units (GRUs). These variants use gates to control the flow of information through the network, making it easier to preserve information from earlier time steps and prevent the vanishing gradient problem [Hochreiter 1998].

2.4.1 Long Short-Term Memory (LSTM)

A Long Short-Term Memory (LSTM) network is a type of recurrent neural network (RNN) that is widely used for processing sequential data [Hochreiter and Schmidhuber 1997]. LSTMs were introduced to address these issues and are designed to be able to store and process information over longer sequences. They achieve this by using a series of gates that control the flow of information into and out of the hidden state. The gates are designed to allow information to be forgotten over time if it is no longer relevant and to prevent the gradients from vanishing or exploding [Hochreiter 1998].

An LSTM network consists of a series of LSTM cells, where each cell takes in an input, the hidden state from the previous time step, and the cell state from the previous time step, and outputs a hidden state and a cell state. The hidden state and the cell state are both used to make predictions in an LSTM. The hidden state is updated at each time step based on the input at that time step and the previous hidden state, while the cell state is updated based on the input at that time step and the previous cell state. The hidden state can be thought of as a summary of the information that has been seen so far in the input sequence, while the cell state can be thought of as the memory of the LSTM that retains information over long periods of time. The hidden state can be used to make predictions about the next output at each time step, while the cell state is used to maintain long-term dependencies in the input sequence.

The gates within each cell are responsible for controlling the flow of information into and out of the cell state. There are three types of gates in an LSTM cell: the input gate, the forget gate, and the output gate. The input gate determines which values from the current input will be stored in the cell state. The forget gate determines which values from the previous cell state will be forgotten, and the output gate determines which values from the cell state will be passed to the hidden state. The gates are implemented as sigmoid or hyperbolic tangent neural networks, where the values of the gates are between 0 and 1. The standard LSTM architecture is depicted in Figure 2.5, and uses the following notation:

- X : is an entire input trajectory

- x_t : is the input at time step t ($x_t = X[t]$)
- h_t : is the hidden state vector also known as the output vector of the LSTM unit at time step t
- c_t : is the cell state vector at times step t
- F_t : forget gate's activation vector at times step t
- I_t : input/update gate's activation vector at times step t
- O_t : output gate's activation vector at times step t

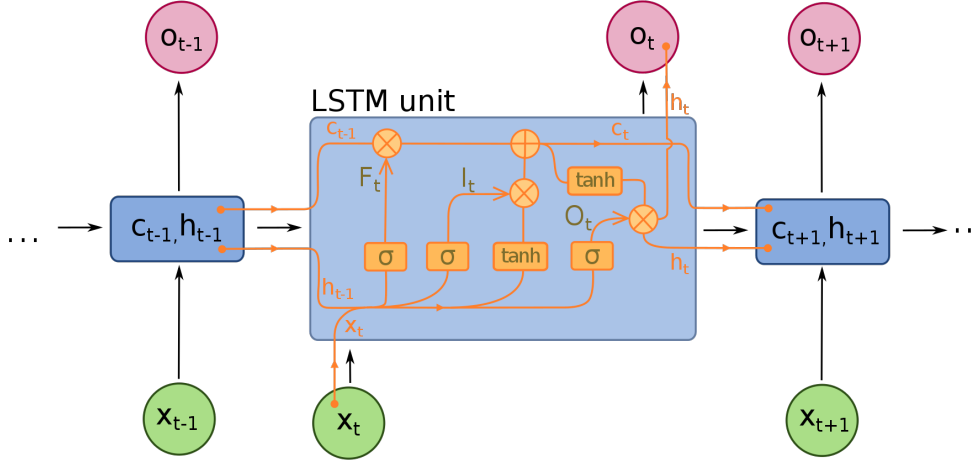


Figure 2.5: Standard LSTM architecture. Here X is an entire input trajectory, x_t is the input at time step t ($x_t = X[t]$), h_t is the hidden state vector and c_t is the cell state vector. F, I, O are the vectors associated with the forget, update and output gates respectively.

Using the aforementioned notation we summarise the forward and back-propagation processes as follows [Hochreiter and Schmidhuber 1997]:

Forward propagation:

- At each time step t , the LSTM takes in an input vector x_t , the previous hidden state h_{t-1} , and the previous cell state c_{t-1} , and produces a new hidden state h_t and a new cell state c_t .
- The input vector x_t is combined with the previous hidden state h_{t-1} to form an input gate activation vector I_t , and with the previous cell state c_{t-1} to form a forget gate activation vector F_t and an input modulation vector $\tilde{C}_t$.
- The input gate activation vector I_t controls how much of the input vector x_t is passed through to the cell state c_t , while the forget gate activation vector F_t controls how much of the previous cell state c_{t-1} is retained in c_t .
- The input modulation vector $\tilde{C}_t$ is used to update the cell state, and is obtained by applying a hyperbolic tangent activation function to the weighted sum of the input vector x_t and the previous hidden state h_{t-1} .
- The new cell state c_t is obtained by combining the input modulation vector $\tilde{C}_t$ with the previous cell state c_{t-1} using the forget gate activation vector F_t .
- Finally, the output gate activation vector O_t is obtained by applying a sigmoid activation function to the weighted sum of the input vector x_t and the previous hidden state h_{t-1} , and the new cell

state c_t is passed through a hyperbolic tangent activation function to produce the new hidden state h_t .

Backward propagation:

- The gradients of the output loss with respect to the output gate activation vectors, the new hidden states, and the new cell states are computed.
- These gradients are used to compute the gradients of the loss with respect to the output modulation vector, the input gate activation vector, the forget gate activation vector, and the input vector.
- The gradients of the loss with respect to the cell state are computed by combining the gradients from the output modulation vector and the forget gate activation vector, and these gradients are backpropagated through the hyperbolic tangent activation function to compute the gradients with respect to the weighted sum of the input vector and the previous hidden state.
- Finally, the gradients with respect to the input vector, the previous hidden state, and the previous cell state are computed by combining the gradients from the input gate activation vector and the forget gate activation vector, and these gradients are backpropagated through the weight matrices to compute the weight gradients.

In a time series prediction task, the final hidden state can be used to predict the next value in the time series, while the cell state can be used to maintain long-term dependencies in the time series.

2.4.2 Gated Recurrent Unit (GRU)

Another variation of an RNN is a GRU, which stands for Gated Recurrent Unit, which similarly to an LSTM uses a gating mechanism to combat vanishing and exploding gradients [Dey and Salem 2017]. A GRU has two gates: a reset gate and an update gate, as seen in Figure 2.6. The reset gate determines how much of the previous hidden state is forgotten, and the update gate determines how much of the previous hidden state is used to update the current hidden state.

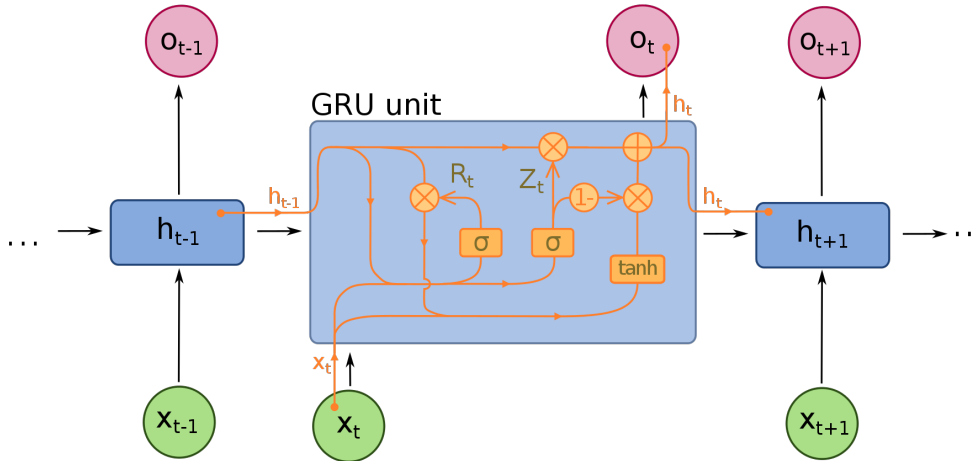


Figure 2.6: Standard GRU architecture. Here X is an entire input trajectory, x_t is the input at time step t ($x_t = X[t]$) and h_t is the hidden state vector. R , Z , O are the vectors associated with the reset, update and output gates respectively.

The forward propagation step is outlined below using the following notation [Chung et al. 2014]:

- X : is an entire input trajectory

- x_t : is the input at time step t ($x_t = X[t]$)
- h_{t-1} is the previous hidden state
- h_t is the current hidden state
- h'_t is the candidate hidden state
- W_r : weights associated with the input for the reset gate
- W_z : weights associated with the input for the update gate
- U_r : weights associated with the previous hidden state for the reset gate
- U_z : weights associated with the previous hidden state for the update gate
- b_r : bias term for the reset gate
- b_z : bias term for the update gate
- r_t : output of the reset gate
- z_t : output of the update gate

The forward propagation step requires calculating the current hidden state (h_t) using Equation 2.10 as a weighted sum of the previous hidden state (h_{t-1}) and a candidate hidden state (h'_t), where the weights are determined by the reset and update gates. The candidate hidden state is computed using Equation 2.9 with the current input and the previous, “resetted” hidden state. Formally, the reset gate r_t and update gate z_t outputs at time step t are computed using Equations 2.7 and 2.8. In a GRU, the reset gate output r_t is a vector that determines how much of the previous hidden state h_{t-1} should be forgotten or remembered [Chung *et al.* 2014]. It takes as input the current input x_t and the previous hidden state h_{t-1} , and outputs a value between 0 and 1 for each element in the hidden state vector. A value of 0 means that the corresponding element in the previous hidden state should be completely ignored, while a value of 1 means that the corresponding element should be fully used. The reset gate allows the model to selectively reset parts of the hidden state that are no longer relevant to the current input, while retaining useful information from previous time steps. The update gate output z_t is a value between 0 and 1 that determines how much of the previous hidden state h_{t-1} should be used to compute the current hidden state h_t [Chung *et al.* 2014]. When the update gate activation is close to 1, it means that the model should rely heavily on the previous hidden state, and when it is close to 0, it means that the model should mostly rely on the candidate hidden state h'_t computed in Equation 2.9.

$$r_t = \text{sigmoid}(W_r \cdot x_t + U_r \cdot h_{t-1} + b_r) \quad (2.7)$$

$$z_t = \text{sigmoid}(W_z \cdot x_t + U_z \cdot h_{t-1} + b_z) \quad (2.8)$$

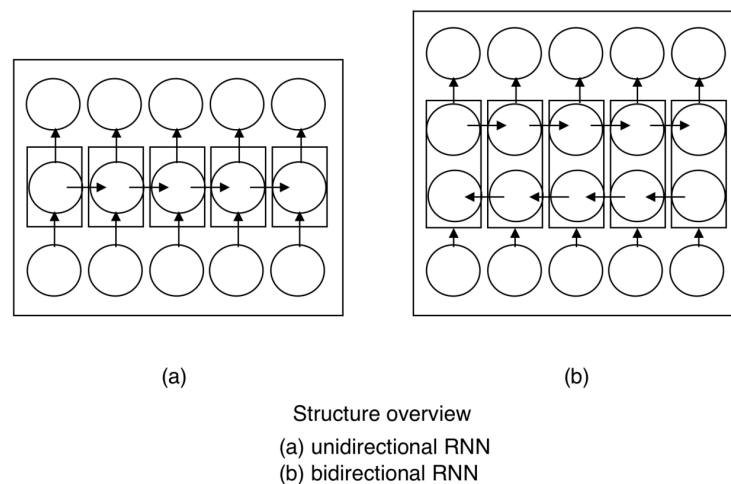
$$h'_t = \tanh(W \cdot x_t + U \cdot (r_t \cdot h_{t-1}) + b) \quad (2.9)$$

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot h'_t \quad (2.10)$$

2.4.3 Bidirectional RNN

Bidirectional recurrent neural networks (BRNN) are a variation to standard RNNs which connect two hidden layers of opposite directions to the same output [Schuster and Paliwal 1997]. By adding this additional connection the output layer can utilise information from the past (backward) as well as the future (forward) states simultaneously. BRNNs, as depicted in Figure 2.7, were introduced to increase the amount of input information available to the network. By processing a sequence in both forward

and backward directions, a BRNN can capture dependencies and patterns that may not be apparent from a unidirectional approach. For example, in natural language processing tasks, a BRNN can use information from both the preceding and following words to better understand the context and meaning of a particular word. This can lead to improved accuracy and performance on various sequence modelling tasks, including speech recognition, sequence labelling, and language translation [Sundermeyer *et al.* 2014].



Similar training algorithms can be used for BRNNs and RNNs since the directional neurons do not interact. However, additional processes are required for back-propagation through time because input and output layer updates cannot be simultaneous. The general training procedure involves passing forward and backward states first during the forward pass, followed by the output neurons. During the backward pass, the output neurons are passed first, followed by the forward and backward states. After completing the forward and backward passes, the weights are updated [Schuster and Paliwal 1997]. Overall all forms of RNNs are powerful tools for automatic advice generation because they are capable of handling sequential data, capturing context, and adapting to changing conditions over time. These are necessary features when looking to generate advice from trajectory information in an online fashion.

An autoencoder is a type of neural network that learns to reconstruct its input ($X_i \in \mathcal{D}$) by compressing it into a lower-dimensional representation (Z_i) and then expanding it back to its original size (X'_i) [Bourlard and Kamp 1988]. An autoencoder is an important type of neural network that is used for unsupervised learning, meaning it does not require labelled data. The autoencoder architecture comprises an encoder network that is interconnected with a decoder network, alongside a loss function that measures the dissimilarity between the input and output data. The encoder and decoder components function as follows:

- The encoder takes in the input and produces a compressed representation, or “code”, of the input. This is usually done by applying a series of linear and nonlinear transformations to the input, gradually reducing its dimensions. The code produced by the encoder is then passed to the decoder.
- The decoder takes the compressed code and reconstructs the original input. This is because the objective of an autoencoder is to learn a compressed representation of the input data in a lower-dimensional space while preserving as much information as possible. By doing so, the decoder can reconstruct the original input data from the compressed code with minimal loss of information. It does this by applying a series of linear and nonlinear transformations to the code, gradually expanding it back to the original size of the input. The output of the decoder should ideally be a close approximation of the original input.

The processing of the encoder and decoder network is illustrated in Figure 2.8

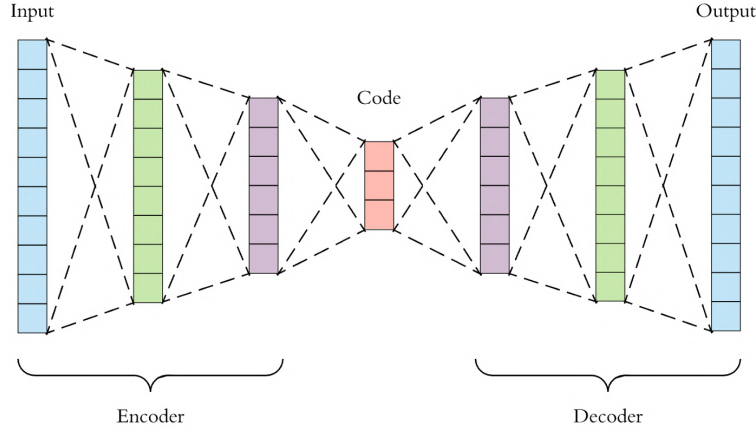


Figure 2.8: Standard Autoencoder Architecture

During training, the autoencoder is optimized to minimize the difference between the input and its reconstructed output. This is typically done using a loss function such as mean squared error (MSE) seen in Equation 2.11 where θ and ϕ are the parameters associated with the weights of the encoder and decoder respectively. Here x_i and x'_i represents the value at the i -th time-step from the original trajectory X and the reconstructed trajectory X' respectively, with m being the length of both.

$$L(\theta, \phi) = \frac{1}{m} \sum_{i=1}^m \|x_i - x'_i\|_2^2 \quad (2.11)$$

By minimizing this loss, the autoencoder learns to encode and decode the input in a way that is most useful for reconstructing the original data. It is primarily used for dimensionality reduction [Wang *et al.* 2016], feature extraction [Meng *et al.* 2017], and anomaly detection [Gong *et al.* 2019]. It is also used as a pretraining step for other neural networks, as the learned encoding can be used as a feature representation for downstream tasks [Sagheer and Kotb 2019].

2.5.1 Variational Autoencoders

A variational autoencoder (VAE) is a type of generative model that can learn to generate new samples similar to a set of training data [Kingma and Welling 2013]. The VAE consists of an encoder network and a decoder network, both of which are typically implemented as neural networks. However, unlike a traditional autoencoder, the VAE is a probabilistic model, which means that it generates a distribution

of encodings for each input data sample. Specifically, the encoder maps the input data to a probability distribution over the latent space, and the decoder samples from this distribution to reconstruct the original data. This allows VAEs to learn a continuous and smooth latent space that can be used to generate new data points by sampling from the learned distribution [Bowman *et al.* 2015].

During training, the VAE learns to regularize the distribution of generated encodings so that it closely approximates a prior distribution, which is usually a standard normal distribution. This is done by minimizing the KL divergence between the encoder’s distribution and the prior distribution as seen in Equation 2.12, which encourages the VAE to learn a smooth and continuous latent space. Here θ and ϕ are the parameters associated with the weights of the encoder and decoder respectively and M corresponds to the number of sampled encodings. Additionally, $D_{KL}(q_\theta(Z_{i,m} | X_i) || p_\phi(Z_i))$ represents the KL divergence between the posterior distribution $q_\theta(Z_{i,m} | X_i)$ of the latent variable Z given an input X_i and the prior distribution $p_\phi(Z_i)$ of Z . In this case, $q_\theta(Z_{i,m} | X_i)$ represents the distribution of the latent variable Z given input x_i , which is modelled by the encoder with parameters θ . Here, m is an index used to represent the m^{th} sampled encoding of Z . The equation represents the average KL divergence loss over all M sampled encodings of Z , which is used to regularize the latent space during training. The lower the KL divergence loss, the closer the learned distribution of Z is to the prior distribution, and hence, the better the model is at generating new data samples from the latent space.

$$L(\theta, \phi) = \frac{1}{M} \sum_{m=1}^M D_{KL}(q_\theta(Z_{i,m} | X_i) || p_\phi(Z_i)) \quad (2.12)$$

The VAE can then be used to generate new data samples by sampling encodings from the prior distribution and passing them through the decoder network. The decoder network maps each “code” to a corresponding output sample, which can be a new image, sound, or any other type of data. Overall, the VAE is a powerful generative model that can learn to generate new data samples with high fidelity to the training data, while also learning a meaningful and continuous latent space that can be used for tasks such as data compression [Graving and Couzin 2020], visualization [Fortuin *et al.* 2018], and manipulation [Hu *et al.* 2019].

2.5.2 Attention

Attention is a mechanism in machine learning and natural language processing that allows a model to selectively focus on certain parts of the input or output sequence [Vaswani *et al.* 2017]. In the context of sequence-to-sequence models such as neural machine translation or text summarising [Ghader and Monz 2017], attention helps the model to align the source and target sequences by assigning a weight to each input token based on its relevance to the current output token.

At a high level, the attention mechanism works as follows:

- The model takes in an input sequence (e.g., a sentence in one language) and produces an encoding of that sequence, typically using a recurrent neural network (RNN) or a transformer [Wolf *et al.* 2020].
- At each step of the decoding process (e.g., generating a word in the target language), the model computes a set of attention weights that reflect the importance of each input token in producing the current output token.
- These attention weights are used to weight the input encoding, producing a context vector that summarizes the relevant parts of the input sequence for the current step of decoding.

- This context vector is then concatenated with the previous output token and fed into the decoder to generate the next output token.

The attention mechanism has become a key component of state-of-the-art natural language processing models, as it enables the model to selectively focus on the most important parts of the input sequence and can improve both the quality and speed of generation. The different types of autoencoders are essential architectures that provide the foundation for the play-style identification approach. This is due to their ability to be trained in an unsupervised manner, without the requirement of labelled data, enabling data compression and making our clustering method viable. The components discussed in these sections influenced the design of our play-style identification model described in Chapter 5.

2.6 Clustering

Sections 2.3, 2.4 and 2.5 focused on the different architectures relevant to the processing of temporal information. The second component of this research is to be able to identify the different play-styles present within a domain. To solve this problem, we implement a clustering-based approach to cluster trajectories with respect to play-styles in an unsupervised fashion. For this reason, a background in clustering is required.

Clustering is a machine learning technique that involves grouping a set of objects or data points into subsets, or clusters, based on their similarity [Milligan and Cooper 1987]. Traditionally clusters have been defined as groups with small distances between cluster members, dense areas of the data space, intervals or particular statistical distributions [Williams 1971]. Clustering can be formalised as an unsupervised learning problem where the goal is to find a partition of the data such that objects within the same cluster are more similar to each other than to objects in different clusters.

Formally, given a set of n data points $X = \{x_1, x_2, \dots, x_n\}$, the goal of clustering is to find a partition $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_k\}$, where k is the number of clusters, and each $\mathcal{P}_i$ is a subset of X such that:

- The union of all the clusters equals the entire data set, i.e., $\mathcal{P}_1 \cup \mathcal{P}_2 \cup \dots \cup \mathcal{P}_k = X$.
- Each cluster is non-empty, i.e., $\mathcal{P}_i \neq \emptyset \forall i$.
- Objects within the same cluster are more similar to each other than to objects in different clusters.

Clustering can be performed using various algorithms such as k-means [Edwards and Cavalli-Sforza 1965], hierarchical clustering [Murtagh and Contreras 2012] and density-based clustering [Kriegel *et al.* 2011]. These algorithms typically involve defining a distance or similarity metric between objects, and then partitioning the data based on this metric [Grabusts 2011].

The resulting clusters can be evaluated using various metrics such as the within-cluster sum of squares (WCSS), silhouette score, or entropy. The analysis plays a vital role in uncovering interpretable properties of the clusters. Cluster analysis is not a deterministic task, but rather an iterative process of knowledge discovery or interactive multi-objective optimisation that involves experimentation and refinement, as noted by Chowdary *et al.* [2014] in their evaluation. Clustering is used in a wide range of applications, including image processing [Coleman and Andrews 1979], text mining [Allahyari *et al.* 2017], bioinformatics [Masood and Khan 2015], and recommendation systems [Sarwar *et al.* 2002]. Overall, clustering is a useful tool for play-style identification in video games, as it can allow for the identification of common patterns and behaviours among players.

2.6.1 K-Means

K-means clustering is a centroid-based clustering method of vector quantization, that aims to partition a set of N data points ($\mathcal{D} = \{X_1, X_2, \dots, X_N\}$) into k clusters, where k is a pre-specified number [Edwards and Cavalli-Sforza 1965].

The algorithm iteratively assigns each data point to the nearest cluster center and then updates the cluster centers based on the mean of the data points in each cluster. Due to its ubiquity, it is often called “the k-means algorithm” [Edwards and Cavalli-Sforza 1965]. Given an initial set of k -means also known as centroids ($\{\mathcal{C}_1^{(1)}, \dots, \mathcal{C}_k^{(1)}\}$) the algorithm proceeds by alternating between two steps:

- Assignment step: Assign each data point to the cluster whose mean has the least squared Euclidean distance; this is intuitively the “nearest” mean. Here i is used to identify a particular cluster.

$$\mathcal{P}_i^{(t)} = \left\{ X_p : \|X_p - \mathcal{C}_i^t\|^2 \leq \|X_p - \mathcal{C}_j^t\|^2 \forall 1 \leq j \leq k \right\} \quad (2.13)$$

where each X_p is assigned to exactly one partition $\mathcal{P}_i^{(t)}$. In general, Equation 2.13 compares the distance from X_p to its current nearest mean $\mathcal{C}_i^t$ with the distance to all other means $\mathcal{C}_j^t$.

- Update step: Calculate the new means (centroids) of the observations in the new clusters.

$$\mathcal{C}_i^{t+1} = \frac{1}{|\mathcal{P}_i^{(t)}|} \left[\sum_{X_j \in \mathcal{P}_i^{(t)}} X_j \right] \quad (2.14)$$

The k-means algorithm aims to minimise the sum of squared distances between each data point and its assigned cluster center. This objective function is also known as the within-cluster sum of squares (WCSS) [Edwards and Cavalli-Sforza 1965]. The algorithm has converged when the assignments no longer change, however, it does not guarantee finding the optimum [Hartigan and Wong 1979].

The resultant cluster assignments can be used to visualise the partitioning of data by utilising a mathematical technique called Voronoi diagrams or Voronoi tessellations [Ying *et al.* 2015]. Specifically, Voronoi diagrams are a partitioning of a given space into regions based on the distance to a specific set of objects in that space. Each region, called a Voronoi cell or Voronoi region, consists of all the points in space that are closer to a particular object than to any other object in the set. Formally, given a set of N points in a d -dimensional space, denoted as $\mathcal{D} = \{X_1, X_2, \dots, X_N\}$, the Voronoi cell of a point X_i is defined as the set of all points in space that are closer to X_i than to any other point in $\mathcal{D}$. Mathematically, the Voronoi cell of X_i can be expressed by Equation 2.15.

$$\mathcal{V}_i = \{G \in \mathbb{R}^d \mid \text{dist}(G, X_i) \leq \text{dist}(G, X_j) \text{ for all } j \neq i\} \quad (2.15)$$

where $\text{dist}(X, G)$ denotes the Euclidean distance between points X and G in the d -dimensional space. Therefore, the Voronoi diagram is the collection of all Voronoi cells associated with the set of N points $\mathcal{D}$ as seen in Figure 2.9.

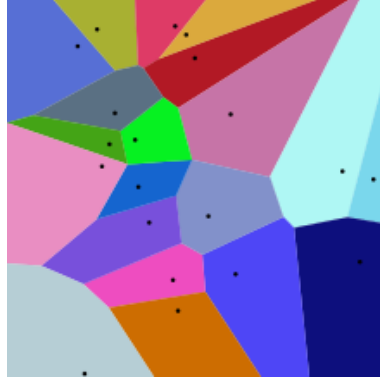


Figure 2.9: 20 points and their Voronoi cells

2.6.2 GMM

Gaussian Mixture Models (GMM) are a probabilistic model used for clustering, which assumes that the underlying data distribution is a mixture of multiple Gaussian distributions [He *et al.* 2010]. A GMM is a generative model that models the data distribution as a weighted sum of k -Gaussian components, where k is the number of clusters. This is formally shown in Equation 2.16 where the parameters are $\Theta = (\alpha_1, \dots, \alpha_k, \theta_1, \dots, \theta_k)$ such that $\sum_{i=1}^k \alpha_i = 1$ and each p_i is a Gaussian density function parameterized by θ_i .

$$P(x | \Theta) = \sum_{i=1}^k \alpha_i p_i(x | \theta_i) \quad (2.16)$$

To fit the GMM model to the data, we first initialize the parameters of the k Gaussian components (mean, covariance, and weight) randomly. Here we look to find Θ such that $p(X | \Theta)$ is a maximum otherwise known, as the maximum likelihood (ML) [Myung 2003]. The log-likelihood function is typically introduced to estimate Θ , defined in Equation 2.17.

$$\mathcal{L}(\Theta) = \log P(X | \Theta) = \log \prod_{i=1}^m P(x_i | \Theta) = \sum_{i=1}^m \log \left[\sum_{j=1}^k \alpha_j p_j(x_i | \theta_j) \right] \quad (2.17)$$

Then we use the Expectation-Maximization (EM) algorithm to estimate the parameters iteratively [Bishop and Nasrabadi 2006]. In the E-step, we compute the posterior probability of each data point belonging to each of the k Gaussian components, based on the current parameters. This step involves calculating the likelihood of the data point given each Gaussian component and normalizing the probabilities across all the components. In the M-step, we update the parameters of each Gaussian component based on the posterior probabilities computed in the E-step. Specifically, we update the mean and covariance of each Gaussian component based on the weighted sum of the data points, where the weight is the posterior probability of the data point belonging to that component. We also update the weight of each Gaussian component based on the sum of the posterior probabilities of all data points. The EM algorithm iterates between the E-step and M-step until the parameters of the Gaussian components no longer change significantly between iterations. Once we have the fitted GMM model, we can use it for clustering by assigning each data point to the Gaussian component with the highest posterior probability.

GMM has several advantages over other clustering algorithms, including its ability to model non-spherical clusters and estimate the uncertainty in cluster assignment using posterior probabilities. However, GMM is sensitive to the initialisation of the parameters and may converge to a local optimum, so it is important to run the algorithm multiple times with different initializations to ensure a good solution [Fraley and Raftery 1998]. Additionally, this is done to ensure that the degenerate infinite likelihood solution is avoided, where a single point is covered by a single component with infinite height and zero variance.

2.6.3 Self-organising maps

A self-organizing map (SOM) or self-organizing feature map (SOFM) is a type of NN that is trained to produce a low-dimensional, discretized representation of the input space of the training samples. It consists of a layer of neurons arranged in a grid-like topology, where each neuron is associated with a weight vector that defines its position in the feature space. So far, the neural networks that have been discussed (Sections 2.3, 2.4) have utilised error-correction learning, such as back-propagation with gradient descent. In contrast, self-organizing maps (SOMs) employ a different technique known as competitive learning [Rumelhart and Zipser 1985]. The main difference between error-correction learning, such as back-propagation, and competitive learning used in SOMs is the objective of the learning process.

In error-correction learning, the goal is to minimize the difference between the predicted output and the true output, which is usually represented as a loss function. The algorithm updates the weights of the neural network to minimize this loss function using methods such as gradient descent. This type of learning is used for supervised learning tasks, where the network is trained on labelled data to make predictions on new, unseen data [Amari 1993].

On the other hand, competitive learning, such as that used in SOMs, is an unsupervised learning method where the goal is to find the structure in the input data. In competitive learning, the neurons compete with each other to become the “winner” by being the closest to the input data point. The weights of the winning neuron and its neighbours are then updated to move them closer to the input data point, and the other neurons remain unchanged. The objective is to create a low-dimensional representation of the high-dimensional input space by grouping similar inputs together in the same neuron [Rumelhart and Zipser 1985].

The goal of learning in SOMs is to cause different parts of the network to respond similarly to certain input patterns. This approach to learning is partially motivated by the human brain where different types of sensory information are handled by different components of the cerebral cortex [Haykin 1994]. SOMs are a data-driven approach and therefore, for effective training, the network requires a large number of example vectors. The process of training is outlined as follows with a formal description seen in Algorithm 1:

- An example is fed to the network and the Euclidean distance to all weight vectors is computed (line 4).
- The best matching unit (BMU) is determined. This unit is selected by comparing the example vector to the weight vector of all other neurons. The neuron whose vector is most similar to the input is, therefore, the BMU (line 5).
- The weights of both the BMU and the neurons in the neighbourhood of the BMU are adjusted towards the input vector (lines 6 to 10).
- Through the training process, the magnitude in which the nodes are adjusted will decrease with time and with grid-distance from the BMU (line 12).

Algorithm 1 Self-Organizing Map Training

- 1: Initialize the SOM weight vectors W to random values in the feature space
 - 2: Set the current iteration t to 0
 - 3: **while** stopping criterion not met **do**
 - 4: Randomly select an input data point $X \in \mathcal{D}$
 - 5: Find the neuron η^* with the closest weight vector to X : $\eta^* = \underset{j}{\operatorname{argmin}} [\operatorname{dist}(X, w_j)]$
 - 6: Update the weight vectors of the neurons (η) in the neighbourhood of η^* :
 - 7: **for** $i = 1$ to τ_1 **do**
 - 8: **for** $j = 1$ to τ_2 **do**
 - 9: Compute the neighbourhood function $\mathcal{N}_{i,j}(t)$
 - 10: Update the weight vector: $w_{i,j}(t+1) = w_{i,j}(t) + \alpha_t \cdot \mathcal{N}_{i,j}(t) \cdot (X - w_{i,j}(t))$
 - 11: Increment the iteration: $t \leftarrow t + 1$
 - 12: Update the learning rate: $\alpha_t \leftarrow \alpha_0 \cdot e^{-t/N}$
-

In this algorithm, τ_1 and τ_2 are the dimensions of the SOM grid, $\operatorname{dist}(X, w_j)$ is the Euclidean distance between the input data point X and the weight vector w_j , α_t is the learning rate at iteration t , α_0 is the initial learning rate, and N is a parameter that controls the rate of decrease of the learning rate. The neighbourhood function $\mathcal{N}_{i,j}(t)$ is typically defined as a Gaussian function centered at the winning neuron η^* , with a width that decreases over time. The stopping criterion can be based on various metrics, such as the number of iterations, the magnitude of weight vector updates, or the change in the quantization error. The general concept behind self-organising maps is that during the training process, the weights of the entire neighbourhood are updated in the same direction as seen in Figure 2.10. This is because similar items tend to excite adjacent neurons. As a result, SOMs create a semantic map where similar samples are grouped closely together, while dissimilar samples are placed further apart.

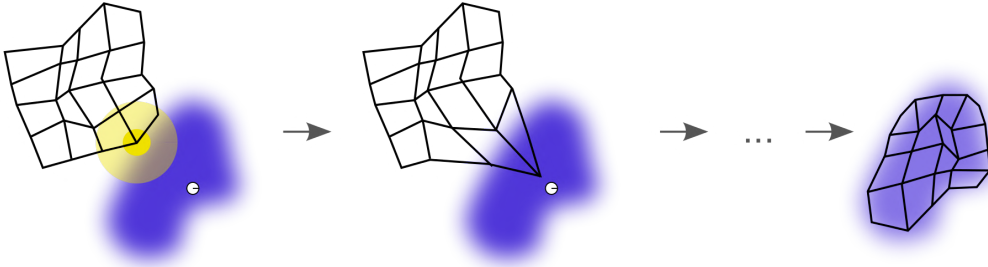


Figure 2.10: Illustration of SOM training procedure. The blue blob represents the distribution of the training data, while a small white disc represents the current input data point drawn from that distribution. Initially, the nodes of the SOM are randomly positioned in the data space, as shown on the left. During each iteration of the training process, the SOM node that is closest to the current training datum is identified and highlighted in yellow. This node is moved towards the input data point, and its neighbouring nodes on the grid are also updated to a lesser extent. After many iterations, the grid tends to approximate the data distribution, as shown on the right.

This type of learning is used for clustering [Vesanto and Alhoniemi 2000], visualisation [Vesanto 1999], and dimensionality reduction tasks [Huang *et al.* 2019], where the structure in the input data is not known [Kohonen 1990]. The components discussed in these sections influenced the design of our play-style identification model described in Chapter 5.

2.7 Markov Decision Process

Although clustering can help us identify general trends such as play styles, it falls short in allowing us to develop a model that can accurately predict behaviours at the level of individual actions. In contrast, comprehending the notion of Markov decision processes (MDPs) is crucial for generating personalised advice automatically. This is because MDPs provide a mathematical framework for modelling decision-making problems in uncertain environments [Bellman 1957]. This framework is represented by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ where:

- $\mathcal{S}$ is a finite set of states that the system can be in.
- $\mathcal{A}$ is a finite set of actions that the decision-maker can take.
- P is a state transition probability function that specifies the probability of moving from one state to another state when an action is taken. It is defined as:
 - $P(s'|s, a) = \Pr[s_{t+1} = s' | s_t = s, \mathcal{A}_t = a]$ where $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$.
- R is a reward function that assigns a scalar reward to each state-action pair. It is defined as:
 - $R(s, a) = \mathbb{E}[R_{t+1} | s_t = s, a_t = a]$ where R_{t+1} is the immediate reward received after taking action a in state s .
- γ is a discount factor that determines the relative importance of immediate rewards versus future rewards. It is a scalar value between 0 and 1, and is used to discount future rewards. The discount factor is typically used to ensure convergence of the value function, which is a key component of many algorithms used to solve MDPs.

The goal of an MDP is to find an optimal policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$ that maximises the expected cumulative reward over time as defined in Equation 2.18:

$$\pi^*(s) = \operatorname{argmax}_a Q^*(s, a) \quad (2.18)$$

where $Q^*(s, a)$ is the optimal action-value function, which is defined as the maximum expected cumulative reward that can be obtained by taking action a in state s , and then following the optimal policy from the resulting state. It is given by the Bellman optimality defined in Equation 2.19:

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s'} P(s' | s, a) \max_{a'} Q^*(s', a') \quad (2.19)$$

Here $\max_{a'} Q^*(s', a')$ is the maximum expected cumulative reward that can be obtained by taking any action a' in the resulting state s' , and then following the optimal policy from that state. The solution to the Bellman optimality equation gives the optimal Q^* function, which can be used to obtain the optimal policy π^* . Alternatively, we can define a state-value function in Equation 2.20, denoted as $V^\pi(s)$, which represents the expected cumulative reward starting from state s under a policy π .

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid s_0 = s \right] \quad (2.20)$$

In Equation 2.20 R_{t+1} is the immediate reward received at time $t + 1$, γ is the discount factor, and $\mathbb{E}_\pi$ denotes the expectation under the policy π . Similarly to Equation 2.18 we can find an optimal policy π^* defined in Equation 2.21 using an optimal $V^*(s)$.

$$\pi^*(s) = \operatorname{argmax}_a \left\{ \sum_{s'} P(s'|s, a) (R(s'|s, a) + \gamma V^*(s')) \right\} \quad (2.21)$$

Formally, the process of solving an MDP involves searching for the optimal policy π^* , the optimal action-value function Q^* , or the optimal state-value function $V^*(s)$. This can be achieved using various algorithms, including policy iteration [Lagoudakis and Parr 2003], Q-learning [Watkins and Dayan 1992], and value iteration [Bellman 1957]. Value iteration, Q-learning, and policy iteration are all algorithms which fall under reinforcement learning to learn optimal decision-making policies in an environment.

2.8 Reinforcement Learning

Reinforcement learning is an approach in machine learning that enables an agent to learn through interactions with an environment [Sutton and Barto 2018]. At the core of reinforcement learning lies MDPs, which provide a mathematical framework for defining problems. The uncertainties in MDPs arise from the fact that the probabilities or rewards are unknown. RL algorithms serve as a general method for handling such Markovian problems. RL algorithms work towards achieving a complex objective or optimising a metric by iteratively improving over many steps. In a gaming environment, the objective could be to maximise the number of points scored, for example. The key challenge that RL addresses is the correlation between immediate actions and the delayed rewards that they produce. Much like humans, RL algorithms often need to wait for some time to see the consequences of their decisions. To incentivise desirable behaviours, RL algorithms provide rewards while penalising undesirable ones. This approach ensures that the agent learns through trial and error, receiving feedback for its actions in the form of positive or negative rewards. By doing so, the algorithm can learn a policy that maximises the expected cumulative reward over the long term. RL is used in many applications, including robotics, game-playing, and autonomous control systems.

In reinforcement learning, an agent interacts with its environment over a series of discrete time steps as seen in Figure 2.11. At each time step t , the agent receives the current state of the environment, denoted by s_t , along with a reward r_t associated with the previous action. The agent then selects an action $a_t \in \mathcal{A}$ from the set of available actions, which is subsequently sent to the environment. The environment responds by transitioning to a new state s_{t+1} and generating a reward r_{t+1} associated with the transition (s_t, a_t, s_{t+1}) . The goal of the agent is to maximise the cumulative reward over a sequence of time steps.

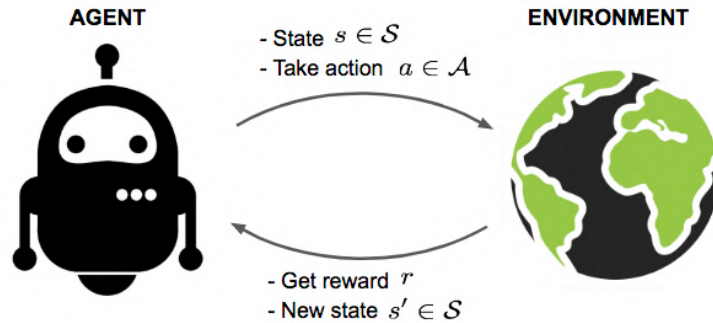


Figure 2.11: Illustration of the cycle of interaction between an RL agent and the environment ³

Although RL has shown impressive success in various applications, such as game-playing, robotics, and recommendation systems [Kaelbling *et al.* 1996] it often requires a large number of interactions with the environment to learn an effective policy [Botvinick *et al.* 2019]. This can be time-consuming and expensive, particularly for real-world applications. Additionally, RL algorithms can struggle to generalise to new environments or tasks, particularly when the training data is limited or the new tasks are significantly different from the training tasks [Cobbe *et al.* 2019]. We utilise reinforcement learning in the generation of our Gridworld datasets described in Section 4.5.

2.8.1 Q-learning

Q-learning is a model-free reinforcement learning algorithm which is able to learn an optimal policy which can be utilised to indicate what is the best action to perform in different situations [Watkins and Dayan 1992]. It uses the Bellman equation previously defined in Equation 2.19 to iteratively estimate the optimal action-value function $Q^*(s, a)$ of an agent in an environment. The “Q-value” represents the expected cumulative reward an agent will receive by taking a specific action a in a particular state s and following an optimal policy thereafter. The optimal policy is the policy that maximises the expected cumulative reward.

The Q-learning algorithm updates the Q-values for each state-action pair (s, a) based on the observed reward and the estimated optimal Q-values for the next state-action pairs. The Q-learning update equation can be written as:

$$Q(s, a) = Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right] \quad (2.22)$$

where:

- $Q(s, a)$ is the Q-value of the state-action pair (s, a)
- α is the learning rate, which determines to what extent newly acquired information overrides old information. A factor of 0 makes the agent learn nothing (exclusively exploiting prior knowledge), while a factor of 1 makes the agent consider only the most recent information (ignoring prior knowledge to explore possibilities)
- r is the reward received for taking action a in state s
- γ is the discount factor, which controls the weight given to future rewards. A factor of 0 will make the agent short-sighted by only considering current rewards, while a factor approaching 1 will make it strive for a long-term high reward.
- s' is the next state reached by taking action a in state s
- a' is the optimal action in state s

Formally, the entire learning process, as described in Algorithm 2, involves iteratively updating the Q-values until they converge to $Q^*(s, a)$, which represents the optimal table of Q-values. This iterative update is performed over a number of episodes. The optimal policy is thus to always perform the action which corresponds to the highest Q-value in a particular state.

³<https://medium.com/analytics-vidhya/reinforcement-learning-machines-learning-by-interacting-with-the-world-64e5862dbf19>

Algorithm 2 Q-learning Algorithm

Initialise Q-value table $Q(s, a)$ for all states s and actions a
for each episode **do**
 Initialise starting state s
 while s is not terminal **do**
 Choose action a using an exploration/exploitation policy based on $Q(s, a)$
 Take action a and observe reward r and new state s'
 Update Q-value of current state-action pair using the formula:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

Update current state to new state s'

Exploration and exploitation are two important concepts in reinforcement learning which impacts action selection as seen in Algorithm 2. Exploration refers to the process of selecting actions that are not necessarily optimal in order to gain more information about the environment. By exploring, an agent can learn about the rewards associated with different actions and improve its policy over time. Exploitation, on the other hand, involves selecting actions that are believed to be optimal based on the current policy. By exploiting, an agent can maximise its expected reward in the short term. In Q-learning, a balance between exploration and exploitation is achieved by using an epsilon-greedy strategy. This means that with probability epsilon, the agent selects a random action (exploration), while with probability 1-epsilon, it selects the action with the highest Q-value (exploitation). The value of epsilon is gradually reduced over time, allowing the agent to shift from exploration to exploitation as it becomes more confident in its estimates of the Q-values [Coggan 2004].

One of the major advantages of this approach, to solving for optimal policies, is that it does not require a model of the environment. Q-learning is able to handle problems with stochastic transitions and rewards in part because of the use of discounted rewards. The discount factor allows the agent to assign more importance to immediate rewards and less importance to uncertain future rewards. Additionally, the agent is able to learn from the stochasticity of the environment through the process of Temporal Difference Learning, where it updates its estimates of the value of each action based on the difference between the observed reward and the expected reward, and adjusts its estimates accordingly. Q-learning is, therefore, able to obtain optimal policies for any problem which can be represented as a finite MDP [Melo *et al.* 2008].

2.8.2 Deep Q Networks

Recent breakthroughs in computer vision and speech recognition have relied on efficiently training deep neural networks on very large training sets. The most successful approaches are trained directly from the raw inputs, using lightweight updates based on stochastic gradient descent. By feeding sufficient data into deep neural networks, it is often possible to learn better representations than handcrafted features [Krizhevsky *et al.* 2012]. Deep Q Networks (DQNs) are a type of deep reinforcement learning algorithm used for solving complex decision-making tasks [Mnih *et al.* 2013]. DQNs use a combination of neural networks and Q-learning to learn an optimal policy for an agent in an environment. This variation of Q-learning replaces the traditional state-action table with a neural network to handle large-scale tasks, which may involve a vast number of possible state-action pairs. By using a parameterized function, the agent can generalize its Q-value estimates to unseen state-action pairs, based on the learned patterns from the training data. This allows the agent to handle large state spaces with high-dimensional inputs,

such as images or raw sensor data. The function approximation approach involves training a neural network to predict the Q-values of state-action pairs, based on the agent's experience. The neural network takes the state as input and outputs the predicted Q-values for all possible actions. During training, the agent interacts with the environment and updates the parameters of the neural network using stochastic gradient descent to minimize the difference between the predicted Q-values and the actual Q-values obtained from the Bellman equation (Equation 2.19).

Along with the same standard Q-learning components described in Section 2.8.1, DQNs also utilise a technique known as experience replay [Lin 1993]. Here the agent's experiences at each time-step, $\epsilon_t = (s_t, a_t, r_t, s_{t+1})$ are stored in a data-set $D = \epsilon_1, \dots, \epsilon_N$, which is pooled over many episodes into a replay memory. Algorithm 3 represents the process of training a Deep Q Network. During the inner loop of the algorithm, a Q-learning style update is applied to samples of experience, which are drawn randomly from the pool of stored samples. After performing experience replay, the agent selects and executes an action according to an epsilon-greedy policy.

Algorithm 3 Deep Q-learning with Experience Replay

```

1: Initialise replay memory  $\mathcal{M}$  to some fixed capacity
2: Initialise action-value function  $Q$  with random weights  $\theta$ 
3: Initialise target action-value function  $\hat{Q}$  with weights  $\theta^- = \theta$ 
4: for episode = 1 to  $\rho$  do
5:   Initialise state  $s_1$ 
6:   for time step  $t = 1$  to  $m$  do
7:     With probability  $\epsilon$  select a random action  $a_t$ , otherwise select  $a_t = \operatorname{argmax}_a Q(s_t, a; \theta)$ 
8:     Execute action  $a_t$  in the environment and observe reward  $r_t$  and next state  $s_{t+1}$ 
9:     Store transition  $(s_t, a_t, r_t, s_{t+1})$  in  $\mathcal{M}$ 
10:    Sample a minibatch of transitions  $(s_j, a_j, r_j, s_{j+1})$  from  $\mathcal{M}$ 
11:    Set  $y_j = r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^-)$ 
12:    Perform a gradient descent step on  $(y_j - Q(s_j, a_j; \theta))^2$  with respect to the network parameters  $\theta$ 
13:  Every  $\varrho$  steps, update the target network:  $\theta^- \leftarrow \theta$ 

```

In this algorithm, ρ is the number of episodes to train, m is the maximum number of time steps per episode, ϵ is the exploration probability, γ is the discount factor, and ϱ is the number of steps between target network updates. $Q(s_t, a_t; \theta)$ and $\hat{Q}(s_t, a_t; \theta^-)$ are the action-value functions with weights θ and θ^- , respectively, and y_j is the target value for the j th transition.

2.9 Imitation Learning

Imitation learning is a type of machine learning where an agent learns to perform a task by imitating the behaviour of an expert [Schaal 1996]. The basic idea behind imitation learning is to learn a policy which maps between sensory input and actions that is similar to the mapping used by the expert.

The process of imitation learning typically involves three main steps:

- **Data collection:** A dataset is collected that contains examples of the expert performing the task. This dataset is used to train the imitation learning algorithm.
- **Learning a policy:** The imitation learning algorithm uses the dataset to learn a policy that maps sensory input to actions. This policy is designed to mimic the behaviour of the expert as closely as possible.

- **Deployment:** The learned policy is deployed in a new environment, where the agent attempts to perform the task based on its sensory input.

Imitation learning can be applied in a variety of settings, including robotics [Schaal 1999], video games [Tencé *et al.* 2010], and natural language processing [Duvallet *et al.* 2013]. One of the main advantages of imitation learning is that it can allow agents to quickly learn complex behaviours without requiring extensive trial and error exploration. However, imitation learning also has some limitations. For example, it can be difficult to ensure that the learned policy is robust to variations in the environment or task. Additionally, imitation learning may not be well-suited to tasks that require creative or novel solutions, as the agent is simply imitating the behaviour of the expert rather than generating new solutions on its own. Imitation learning is particularly useful in scenarios where it is difficult or time-consuming to manually design optimal policies or rules [Schaal 1996]. Instead, by learning from expert demonstrations, imitation learning can automatically generate effective policies that are tailored to specific play-styles. We utilise this feature as it allows for the efficient learning of expert policies.

2.9.1 Behavioural Cloning

Behaviour Cloning (BC) has been utilised for different video game environments using datasets of demonstrations and is a form of imitation learning [Gorman and Humphrys 2007], [Harmer *et al.* 2018], [Jacob *et al.* 2022]. Here an agent learns to exhibit a demonstrated behaviour by mimicking actions $a \in A$ the set of all actions. The idea is to use an expert to demonstrate desired behaviour instead of using a domain expert to design fine-grained rewards. Formally, the network is allowed to learn and adjust its policy to align with the actions demonstrated by an expert. These actions correspond to the actions that an expert demonstrator would take when presented with an observed state $x \in X$, where X represents the entire trajectory. For our models, a trajectory is defined as a sequence of states $(x_0, x_1, \dots, x_m)$ where m is the length of the trajectory as in Equation 2.1. Learning, therefore, requires a dataset $(\mathcal{D} = \{X_0, X_1, \dots, X_N\})$ where N is the number of trajectories) of demonstrated behaviour representative of a policy (π) . A loss function is utilised for behavioural cloning; these include cross-entropy or mean squared error to measure the distance between the predicted and demonstrated actions. The model can then be trained by optimising the following equation:

$$\min_{\theta} \frac{1}{m} \sum_{i=1}^m L(f(x_i; \theta), a_i) \quad (2.23)$$

Here θ represents the parameters of the model, x_i and a_i are the input and target action of the i -th training example, $f(x_i; \theta)$ is the predicted output of the model given input x_i and parameters θ , L is a loss function that measures the difference between the predicted output and the true output, and N is the number of training examples. The objective is to find the set of parameters θ that minimise the average loss over the training set. This reduces the problem of learning sequential action sequences to a highly efficient supervised learning task [Bakker *et al.* 1996]. After training, the optimal parameters (θ^*) of the model can be used to obtain the optimal policy (π^*) as seen in the following equation:

$$\pi^*(x) = f(x; \theta^*) \quad (2.24)$$

We opted to utilise behavioural cloning in our framework due to its wide usage and ability to replicate observed behaviours, which is a key feature of our model as discussed in Chapter 7.

Chapter 3

Related Work

Chapter 2 detailed the architectures and concepts required for the understanding of the underlying components of our solution. In this chapter, we explore several related fields to provide a comprehensive understanding of automatic tailored advice generation and the proposed solutions. In particular, understanding work in the fields of advice generation, play-style identification, player modelling, and imitation learning can be crucial for designing effective automatic tailored advice generation systems with a focus on video games.

Section 3.1 on advice generation provides a starting point for understanding the current state of the art in the field. This includes a review of different approaches to generating advice, such as rule-based systems, collaborative filtering, content-based filtering, and machine learning algorithms.

Section 3.2 on play-style identification is relevant for the understanding of how to personalize advice based on the unique playing style of individual users. This section explores different techniques for identifying play styles, such as clustering algorithms or analysis of game logs, and discusses how play-style identification can be integrated into advice generation systems.

Section 3.3 on player modelling provides insights into how to generate personalised advice based on individual player behaviour. This involves reviewing different player modelling techniques, such as Markov models, Bayesian networks, and decision trees.

Section 3.4 on imitation learning provides a relevant understanding of how to mimic the behaviour of expert players, useful for identifying opportunities for improvement. This includes different approaches to imitation learning, such as inverse reinforcement learning, apprenticeship learning, and deep reinforcement learning.

3.1 Giving Advice

The problem of advice generation is a computational problem that involves generating personalised recommendations or advice for a user based on their preferences, past behaviour, and other relevant factors. The goal of advice generation is to provide helpful and relevant guidance to users in various contexts, such as recommending products, suggesting courses of action, or providing support for decision-making. Researchers from different fields such as psychology [Harvey *et al.* 2000], counselling [Vehvilainen 2001], philosophy [Herrmann 2022], and communication [Garvin and Margolis 2015] have studied this issue. In particular, Garvin and Margolis [2015] investigated the dynamics of giving and receiving advice, providing practical tips on how to communicate effectively and constructively. Our investigation

aims to employ an automated mode of advice generation to differentiate between a user and their corresponding expert, as opposed to prioritising the method of communication.

Automatic advice generation refers to the use of artificial intelligence (AI) and machine learning (ML) techniques to provide personalised advice or recommendations to individuals. It involves the use of algorithms that analyse data about the person seeking advice, such as their preferences, goals, and past behaviour, to generate tailored advice. We look to utilise a suite of different ML approaches to generate advice. The general issue of generating effective advice is analogous to the challenge of interpretability associated with deep learning. Although significant advancements have been made in deploying deep learning algorithms that achieve expert-level performance, in the domains of classification such as [He et al. \[2016\]](#), [Mensink et al. \[2021\]](#) and [Krizhevsky et al. \[2012\]](#) as well as in learning such as [Mnih et al. \[2015\]](#) and [Silver et al. \[2017\]](#), concerns have been raised regarding their lack of interpretability. However, if we are to use such techniques to generate useful applicable advice then we require a better understanding of how and why these algorithms produce the results they do. In regards to the question of “how”, there has been research which has been dedicated to answering this question in the realm of deep image classification [[Fong and Vedaldi 2017](#); [Ribeiro et al. 2016](#); [Simonyan et al. 2013](#); [Zhang et al. 2018](#)]. However, when it comes to deep reinforcement learning networks, there is significantly less literature that delves into explaining the “how” aspect. Despite attempts to explain the behaviour of deep RL agents through the observation of policies, such approaches are becoming less reliable as environmental complexities intensify. Consequently, more sophisticated methods are necessary to generate satisfactory explanations.

One of the approaches which have been used are Jacobian Saliency Maps which were implemented in [Wang et al. \[2015\]](#) to identify which regions of the world returned the strongest signal in Atari2600 games. Saliency has also been applied to natural language, where [Arras et al. \[2017\]](#) trained a word-embedding-based convolutional neural network to be able to identify the importance individual words have on classifying text documents. However, the Jacobian style of saliency generation has been criticised for its generally poor quality [[Kindermans et al. 2017](#)]. This problem was addressed by [Greydanus et al. \[2017\]](#) who implemented a perturbation-based technique which performed well in reducing the noise of the map generated. [Greydanus et al. \[2017\]](#) utilised this visualisation as a way to better understand the Atari agents. A benefit of these less noisy maps is that non-experts can easily identify and understand why the agent is acting a particular way. This approach falls short, however, as it does not provide the observer with any contextual information pertaining to why different regions are important. [Greydanus et al. \[2017\]](#) identified that their approach should be applied in conjunction with other techniques of explanation extraction in order to satisfy human users.

This idea of a suite of machine learning techniques that produce both expert performing models but also more explainable models is echoed by [Gunning \[2017\]](#). Three channels for explainability were identified in their roadmap to Explainable Artificial Intelligence (XAI):

- Deep Explanation – where deep learning techniques are modified to learn explainable features.
- Interpretable Models – which involves techniques to learn more structured, interpretable, casual models.
- Model Induction – which is the technique of inferring an explainable model from any model.

This roadmap outlined the current state of XAI and described the various approaches and aspects a number of institutions will be tackling in the years to come. [Gunning \[2017\]](#) also featured [Ramanishka et al. \[2017\]](#) which addressed the problem of contextual information in saliency maps. They implemented a system called Caption-Guided Visual Saliency which could produce attention values for still images or videos. Their model estimates both a temporal and a spatial mapping between input and output using a

base LSTM encoder-decoder model by recovering the implicit attention from the model. The benefit of their model was twofold; firstly it generated more accurate heat maps while maintaining good captioning and secondly it could be used to understand a wide variety of encoder-decoder architectures. The basis of [Ramanishka et al. \[2017\]](#)’s LSTM encoder-decoder model was built on that of [Venugopalan et al. \[2014\]](#) who implemented an end-to-end deep LSTM model for the video-to-text generation that simultaneously learnt a latent “meaning” state. [Malinowski et al. \[2015\]](#) also utilised the idea of LSTMs in order to generate a system that was capable of answering natural language questions based on images. The results generated from the application of saliency maps could be applied to help identify the characteristics of play-styles that we try to identify in our strategy generator as discussed in Section 4.1. In a parallel fashion, where [Ramanishka et al. \[2017\]](#) and [Venugopalan et al. \[2014\]](#) employed LSTM-based encoder-decoder models for video-to-text generation and latent meaning discovery, our model similarly leverages this architecture to learn patterns and characteristics of gameplay trajectories, ultimately offering tailored guidance to players.

[Gunning \[2017\]](#) also featured [Hu et al. \[2017\]](#) and [Andreas et al. \[2016\]](#) which represent another approach to the problem of explainability making use of the idea of decision trees. [Hu et al. \[2017\]](#) utilised the concept of Neural Module Networks (NMN) which was proposed by [Andreas et al. \[2016\]](#). This NMN architecture makes it possible to answer natural language questions about images using collections of jointly-trained neural “modules”, dynamically composed into deep networks based on linguistic structure. This is a similar problem to that of [Malinowski et al. \[2015\]](#). However, with NMN they showed that it achieves state-of-the-art performance on existing datasets for visual question answering. [Hu et al. \[2017\]](#) noted that although [Andreas et al. \[2016\]](#) learnt a dynamic structure for their NMN it only incorporated a limited form of layout prediction by learning to rerank a list of three to ten candidates, again generated by rearranging modules predicted by a dependency parse. This is a serious limitation because off-the-shelf language parsers are not designed for language and vision tasks and must, therefore, be modified using handcrafted rules that often fail to predict valid layouts [[Johnson et al. 2017](#)]. The approach of [Hu et al. \[2017\]](#) learns to optimize over the full space of network layouts rather than acting as a re-ranker and requires no parser at evaluation time. Their model uses a set of neural modules to break down complex reasoning problems posed in textual questions into a few sub-tasks connected together, and learns to predict a suitable layout expression for each question using a layout policy implemented with a sequence-to-sequence RNN.

An alternate approach to explainability was the idea of rationale generation proposed by [Ehsan et al. \[2018a\]](#) in which they created a system which would generate rationales of autonomous system behaviour as if humans had performed the behaviour. [Ehsan et al. \[2018a\]](#) used neural machine translation to translate internal state-action representations of an autonomous agent into natural language. Their technique required a training corpus that consisted of state-action pairs manually annotated with natural language explanations. They fed their training corpus through an LSTM encoder-decoder model which learned to translate the (state, action, annotation) information into a natural language rationale. [Ehsan et al. \[2018a\]](#) showed that creating rationalizations using neural machine translation techniques produces rationalizations with accuracies above baselines. Additionally, they showed that rationalizations produced using this technique were more satisfying to humans than the other tested alternative means of explanation. The testing and analysis of [Ehsan et al. \[2018a\]](#) were expanded upon in [Ehsan et al. \[2018b\]](#). Here [Ehsan et al. \[2018b\]](#) looked to determine the robustness of the rationale generator by hiding some of the state that was fed into the model. Additionally, they expanded their qualitative analysis where they used human participants to judge the rationale. This time they instructed the participants to rate the rationale using a 5-factor metric. The rationales generated by their technique were judged better than those of random baselines and close to matching the upper bound of human rationales.

All these approaches have focused on generating some understanding or rationale behind the actions produced by the learned model. However, in terms of our goal of creating an advice giver, having the rationale is only half the problem. The other half is to be able to convert that rationale into a form which individual users can utilise to better their performance. In order to accomplish this additional task, we draw on knowledge from the work done in intelligent tutoring systems (ITSs). ITSs are computer-based instructional systems with models of instructional content that specify what to teach, and teaching strategies that specify how to teach [Wenger 2014]. ITSs have moved out of the lab and into classrooms and workplaces where some have been shown to be highly effective [Shute 1991; Graesser *et al.* 2012]. While intelligent tutors are becoming more common and proving to be increasingly effective they are difficult and expensive to build. However, with the progress in cognitive science and the development in the theory of cognition [Anderson 2013], researchers have been able to design better ITSs such as the geometry tutor developed by Anderson *et al.* [1985]. The system worked by generating rules by reducing geometry questions into sub-problems. These rules were used to simulate the sequence of the inferences (correct and incorrect) that students report making in trying to solve a geometry problem. More recently, the adoption of reusable components and learning objects [Ritter *et al.* 2003], as well as the utilisation of community authoring [Aleahmad *et al.* 2008], has become prevalent in facilitating the development of intelligent tutoring systems.

Both ITSs and other rationale generation approaches can benefit from a deeper understanding of the proficiency level of individual students or agents. Similarly, our solution for user advising is enhanced by developing a deeper understanding of the users and environment through play-style identification (Section 3.2).

Alternatively, we can also draw from the work done in the fields of learning where researchers have focused on improving the rate at which agents learn. There is a growing number of techniques that are currently being incorporated to improve RL by leveraging knowledge from outside sources. These techniques include:

- **Transfer Learning:** is when an agent has useful knowledge from a source task to aid its learning in a target task. [Taylor and Stone 2009].
- **Experience Replay:** has a student train on the recorded experiences of a teacher [Lin 1992].
- **Apprentice Learning:** has a student ask a teacher for advice whenever its confidence in a state is low [Clouse 1997; Ho and Ermon 2016].
- **Learning from Demonstration (LfD):** is a paradigm in which a student agent watches another agent (or a human) and learns the teacher’s policy [Argall *et al.* 2009].
- **Inverse Reinforcement Learning (IRL):** is a paradigm in which a student agent observes a teacher and tries to infer the teacher’s reward function [Abbeel and Ng 2004].

A comprehensive review of these approaches in the context of generating expert human-like models is presented in Section 3.4 of this paper. Our model seeks to address this same challenge of learning an advisor tailored to individual users by incorporating a form of imitation learning into its framework.

3.2 Play-style Identification

There are various approaches to play-style identification in different fields such as sports, gaming, and music. One such approach involves the use of statistical methods to identify patterns and trends in data related to gameplay or performance. For example, in sports, statistics such as shots taken, passes completed, and distance covered can be analysed to identify a player’s style of play [Riezebos *et al.* 2021]. In relation to video games Porter *et al.* [2010], Martin [2012], Kartsanis and Murzyn [2016],

Birk *et al.* [2017] and Lindberg and Laine [2016] all looked to use statistical analysis on survey and interview data to ascertain trends in human players. These identified trends have been used to determine relationships between, age, personality and even cognitive ability [Tekofsky *et al.* 2013]. However, statistical approaches are often focused on quantifiable data points, which may not capture the full context of gameplay or performance [Hughes and Bartlett 2002]. Alternatively, expert observers, such as coaches, game analysts or psychologists, have analysed player’s gameplay to identify patterns that define their playing style [Przybylski *et al.* 2010; Phillips *et al.* 2010]. In Particular, Conroy *et al.* [2012] used human observers to identify traits from gameplay to help distinguish between AI and human players. Similarly, Yuda *et al.* [2021] conducted a survey to investigate whether human observers can identify “human-like” traits in the behaviour of game characters. However, all these approaches require careful consideration in both the experimental setup and data analysis. This is due in part to expert observation typically relying on a small sample of data, which may not be representative of a player’s overall playing style or performance [Sampaio and Janeira 2003]. In some cases, players have to be able to identify their own playing style through self-assessment. This approach involves reflecting on one’s own gameplay and analysing their strengths, weaknesses, and tendencies [Veale *et al.* 2019].

However, questions involving biases and subjectivity remain points of concern. In an attempt at a more robust classification Ter Maat [2005] employed a rule-based system to identify play-style to inform an adaptive AI player for Connect Four. Khoshkangini *et al.* [2018] also describes a rule-based system to extract play style dynamically and customize the game on-the-fly for individual players during gameplay sessions. The framework was evaluated using an educational game called Solving the Incognitum¹. Although, the rule-based approaches may be considered more scientifically rigorous than some other methods, they still rely on subjective criteria and may not capture the full range of player behaviours and strategies. This is because they involve explicit criteria for identifying and classifying play styles. Additionally, they may be limited by the specific game or context in which they are applied, and may not be generalizable to other games or situations. Our model aims to sidestep the pitfalls of human bias and the necessity for rule-based systems by integrating principles of machine learning, creating a more unbiased and adaptable approach.

More recently machine learning algorithms have been used to identify patterns and trends in large datasets related to gameplay or performance. Commonly in video games, play-style identification has been approached through player modelling, which is the study of computational models of players in games. This includes the detection, modelling and prediction of human player traits which are manifested through cognitive, affective and behavioural patterns [Yannakakis *et al.* 2013]. The techniques utilised in player modelling usually fall into one of two groups: model-based or model-free.

3.2.1 Model-Based Approaches

In model-based approaches, a player model is built on a theoretical framework whereby the preexisting understanding of the domain is leveraged [Yannakakis *et al.* 2013]. Model-based approaches have been inspired by cognitive frameworks [Anderson *et al.* 1985] as well as general theoretical frameworks of behavioural analysis such as usability theory [Isbister and Schaffer 2008]. Additional examples are models which utilise theories of “fun” [Malone 1980; Koster 2013]. These top-down approaches have also been used to dynamically affect the player experience in terms of difficulty adjustment [Andrade *et al.* 2005; Olesen *et al.* 2008; Spronck *et al.* 2004], or even narrative experience [Lindley 2005]. Although model-based techniques like these are useful, our work seeks to avoid the use of any prior knowledge

¹<https://www.youtube.com/watch?v=PgfuXAdm31o>

of the domain and in essence, learn the heuristic classifier in a model-free setting. Doing so has two benefits, with the first being we are not required to impart any domain-based bias, and the second is the ability to learn a more accurate model than a handcrafted one. Nonetheless, the downside is that we require a greater amount of user gameplay data.

3.2.2 Model-Free Approaches

Model-free approaches refer to the construction of a mapping (model) between (player) input and a player state representation. In this case, there is no pre-existing understanding of the model; rather, it is learned through an iterative process [Yannakakis *et al.* 2013]. To achieve this, observations are collected and analysed to generate a model without strong initial assumptions of its structure. In model-free approaches, we see attempts to model and predict player actions and intentions [Thue *et al.* 2007; Thureau *et al.* 2004]. Thue *et al.* [2007] implemented a system that learns a label for the player representing their style. However, this required the manual annotation of the different “encounters” with their corresponding style. Our proposed approach looks to learn these play-styles in a completely unsupervised setting.

Data mining efforts to identify different behavioural playing patterns within a game have also been implemented using bottom-up approaches [Drachen *et al.* 2009; Weber and Mateas 2009]. Drachen *et al.* [2009] trained emergent self-organising maps on high-level game behaviour data to classify players into four categories representative of their style. Here no pre-existing information was required for the analysis of the data, which also meant that the clusters that were identified were not influenced by external factors. However, this process required identifying the characteristics that resided within the data of each separated cluster. It is this statistical analysis of the separated data that revealed the semantics behind each cluster and allowed for the identification of player archetypes. We perform a similar step, however, the results of this process are dependent on the features present in the original dataset. Additionally, Liu *et al.* [2013] used machine learning algorithms to identify a StarCraft II player from features extracted from game replays. The study found that AdaBoost on a decision tree and Random Forest decision trees perform best for this task. These techniques have been effective, however, these studies have utilised meta-data summaries over raw user data experiences. We aim to achieve the same goal by analysing raw sequence-based trajectory data, but without the need for extensive feature engineering.

3.2.3 Trajectory Clustering

An approach to the analysis of trajectory data is comparing and grouping based on whole or partial trajectory attributes using a similarity measurement. However, it has been demonstrated that clustering using sub-sequences lacked meaning as the generated cluster centres are not dependent on the input [Keogh and Lin 2005]. Additionally, when clustering trajectories, the choice of similarity measure is important as it should consider both the spatial and temporal features [Kisilevich *et al.* 2009]. Common distance measures include Hausdorff distance, Dynamic Time Warping (DTW) [Berndt and Clifford 1994], Euclidean distance and Longest Common Sub-Sequence (LCSS) [Vlachos *et al.* 2002]. The choice of metric is dependent on the structure of the data. DTW and LCSS in particular can measure similarities between varying length trajectories.

Techniques such as k-Means and hierarchical clustering have been utilised to perform the clustering step [Nanni 2002]. However, these techniques tend to be ineffective when input dimensions are high [Steinbach *et al.* 2004]. Additionally, model-based methods which use statistical approaches (COBWEB [Fisher 1987]), Neural Network approaches (ART [Carpenter and Grossberg 1987]), or self-organising maps (SOM [Kohonen 1990]) have been utilised. SOM clustering of time-series features is unsuitable

for trajectories of unequal length, as the dimensions of the weight vectors are fixed [Liao 2005]. Furthermore, data compression techniques utilising Latent Dirichlet Allocation (LDA) have been applied to play-style clustering [Gow *et al.* 2012]. Lange *et al.* [2021] uses data collected from the computer keyboard and mouse telemetry to analyze players’ play-styles and mechanics in detail. The study investigated key control sequences used by players during the game, sequences that distinguish professionals, hardcore amateurs, and casual players, and individual characteristics of players in different groups. The study used Principal Component Analysis and supervised ensembles of Decision Trees with Recursive Feature Elimination (RFE) as a feature selection technique to address these problems. However, by considering timesteps as individual data points, LDA and PCA approaches ignore the structural significance of temporal data. This problem is further exacerbated when you consider the long time horizon inherent in most video games.

Valls-Vargas *et al.* [2015] proposed a player modelling framework that used episodic information and sequential machine learning techniques to capture changes in play style over time. Different trace segmentation strategies were tested for play style prediction, and the framework was evaluated on gameplay data from an interactive learning environment. Valls-Vargas *et al.* [2015] concluded that sequential techniques that incorporated predictions from previous segments outperformed non-sequential techniques. While Valls-Vargas *et al.* [2015] effectively incorporated temporal information and concluded that overly fine or coarse segmentation of traces led to decreased performance, we argue that in video games with long-term dependencies, a more comprehensive solution capable of accurately accounting for long-term dependencies would be preferable.

In addition to these, deep learning techniques have been applied to unsupervised clustering [Xie *et al.* 2016]. Xie *et al.* [2016] used an autoencoder which allowed for the important data compression aspect which was required to make the previously mentioned clustering techniques more feasible. This makes the clustering task easier since clustering can be performed on the encoded data. Xie *et al.* [2016] pre-train a network to minimise the reconstruction loss and then separately minimise the clustering loss. This separate clustering step used the KL-Divergence [Frognier *et al.* 2015] between a target distribution and an estimated distribution generated from the encoded data. More recently, LSTM-autoencoders have been implemented which are capable of handling time-series data effectively [Madiraju *et al.* 2018]. This is a similar approach to Xie *et al.* [2016] with the added LSTM layers to handle time-series data as well as joint minimisation of the reconstruction and clustering aspects. Concurrent autoencoder approaches have been utilised for play-style discovery [Talwadder *et al.* 2022] whereby the latent space was used along with ground truth labels in a supervised fashion. It is not common within video game domains for there to exist labelled trajectories with respect to play-style. Therefore, the utilisation of supervised approaches limits how easily a model can be applied and as a result, we opt to employ an unsupervised approach. We utilise a similar but unsupervised approach to handling our video game-based trajectories which have the added characteristic of being variable in length as well as being multi-dimensional.

There are several limitations associated with latent space clustering techniques. For example, the quality of the latent space representation is heavily dependent on the choice of encoding method used to map the original data to the lower-dimensional space. A sub-optimal encoding can lead to poor clustering results. Tzoreff *et al.* [2018] looked to alleviate such problems by proposing an approach to optimize an auto-encoder using a discriminative pairwise loss function during the pre-training phase. This approach demonstrated improved accuracy of clustering and achieves rapid convergence, even with small networks. Alternatively, Kamnitsas *et al.* [2018] proposed a novel cost function for semisupervised learning of neural networks that encourages compact clustering of the latent space for better separation. Their method combines the benefits of graph-based regularization with efficient inductive inference and can be applied to existing networks without modifications. An additional issue regarding latent space clustering relates to the difficulty in interpreting the results as the clusters obtained may not always be

intuitive or easy to interpret, especially if the data is high-dimensional. [Shen et al. \[2020\]](#) explored how GANs encode different face attributes in the latent space and found that the latent code of well-trained models learns a disentangled representation. They decouple some entangled semantics with subspace projection, achieving more precise control of facial attributes. [Liu et al. \[2019\]](#) proposed methods for mapping and comparing meaningful semantic dimensions within latent spaces, and presented an integrated visual analysis system for this purpose. The system enabled users to discover, define, and verify relationships among data points encoded within latent space dimensions.

Most play-style identification approaches are designed to learn patterns and identify general play-styles across a group of players, without considering the unique playing style of an individual player. However, to provide personalised recommendations and tailored advice, we also need to develop a model that can accurately capture the playing style of a specific person. Such a model would enable us to provide targeted and relevant advice to individual players, which can improve their overall gaming experience and engagement. For this reason Section 3.3 explores work related to learning to predict an individual’s future actions.

3.3 Future State Prediction

3.3.1 Time-Series Forecasting

The standard neural network methods for performing time series prediction include MLP, RBF or Cascade Correlation Models, which use a sliding window of N-tuple inputs to predict a single output target value [\[Gershenfeld and Weigend 1993\]](#). Since these methods are dependent on the size of the rolling window, important long-range dependencies can be forgotten over the course of the full training set. Long Short-Term Memory (LSTM) [\[Hochreiter and Schmidhuber 1997\]](#) solutions are a special form of Recurrent Neural Network’s (RNN) [\[Hua et al. 2019\]](#) which makes use of a component called a “memory block” [\[Hochreiter and Schmidhuber 1997\]](#) which allows it to learn long term dependencies. LSTM prediction models have been utilised in StarCraft2 for learning action selection [\[Vinyals et al. 2019\]](#). We, however, employ an LSTM prediction model to video game-based trajectory data to predict player actions.

3.3.2 Modelling player actions

A common implementation of player modelling is that of modelling a player’s choices (actions) given different scenarios (states) [\[Bakkes et al. 2012\]](#). In its basic form, this consists of a model which indicates the likelihood of any available player action given any state. Traditionally action models were implemented for board game research to improve game tree searches by predicting opponent moves. In this case, the player model was expressed as an evaluation function [\[Cannel and Markovitch 1993\]](#). Similar approaches have been applied to more complex games, however, the increasing sizes of state and action spaces make training accurate models difficult. This has resulted in solutions involving large-scale computing [\[Vinyals et al. 2019\]](#) or utilising abstracted action spaces called “options” [\[Sutton et al. 1999\]](#). Our approach differs from these by using supervised learning instead of reinforcement learning to predict actions similar to [Muñoz et al. \[2013\]](#). When generating advice in a real-time scenario, it is important to have minimal delay in prediction accuracy, meaning that sample efficiency is critical. Sample efficiency refers to the ability of a sampling method to produce a representative sample using the minimum number of observations possible [\[Yarats et al. 2021\]](#). One potential method to enhance sample efficiency is to make use of pre-training, which involves training a model on a large dataset to learn general features and patterns that can later be fine-tuned on a smaller dataset for a specific task [\[Zoph et al. 2020\]](#). In a manner analogous to this, our model utilizes an initial offline learning strategy, supplemented by a subsequent online learning phase, to enhance its understanding of individual be-

behaviour and improve its predictive accuracy. [Albrecht and Stone \[2018\]](#) present an overview of current problems in the realm of player modelling. In particular, it mentions “Modelling changing behaviours” which is a notable component of our model as we look to utilise the temporal nature of gameplay to better identify a user’s play-style. Player modelling is closely related to the field of opponent modelling which concentrates on understanding and predicting the behaviour of opponents or adversaries in a game or competitive setting.

3.3.3 Opponent Modelling

The history of opponent modelling research traces back to [Von Neumann and Morgenstern \[2007\]](#) who had previously introduced game theory. Initially designed for zero-sum games, game theory concepts have since been applied to diverse scenarios, giving rise to various fields, including opponent modelling. Over time, the term “opponent modelling” has evolved to encompass situations that aren’t strictly adversarial, often involving elements of cooperation for mutual benefit [[Baarslag et al. 2016](#)]. [Nashed and Zilberstein \[2022\]](#) presents a comprehensive overview of existing opponent modelling techniques for adversarial domains, many of which must address stochastic, continuous, or concurrent actions, and sparse, partially observable payoff structures. These methodologies share resemblances with our approach, albeit our emphasis lies in modelling user actions within single-player domains, as opposed to concentrating on opponents.

3.4 Game Modelling

The research on learning to generate policies which exhibit human-like behaviour can be roughly categorised into two groups based on whether they are data-based or reward-based. The reward-based approaches rely on optimising a fitness or reward based function which a Genetic Learning (GA) or Reinforcement Learning (RL) algorithm can use for optimisation. The most straightforward data-based approach is Imitation Learning (IL), which uses supervised learning to predict and perform actions with the highest predicted probability.

3.4.1 Reward-based

A common reward based technique is called reward shaping. Here the agent is provided with additional shaping rewards that come from a deterministic function $F : S \times A \times S \rightarrow \mathbb{R}$. These additional reward terms are used to aid in convergence towards a desired behaviour where S is the finite set of states, and A is the finite set of all actions [[Marom and Rosman 2018](#)]. This idea of reward shaping has been used to train a set of human-like bots with differing styles [[Arzate Cruz and Ramirez Uresti 2018](#)]. Here the standard reward function R was augmented to resemble $R' = R + F$ where F encouraged the play-style-specific behaviours they desired. This resulted in agents with 3 varying styles in the game of Street Fighter 5. [Khalifa et al. \[2016\]](#) demonstrated a similar approach to encouraging bots to act in a certain style by augmenting the standard Monte Carlo Tree Search algorithm. Here an additional term was added to the Upper Confidence Bound equation which governs which nodes are explored, in order to select more human-like actions. The limitation of these approaches is the design process of the reward functions not guaranteeing the desired results. [Sephton \[2016\]](#) explored an alternate design-based approach using several techniques of modifying Monte Carlo Tree Search (MCTS) to create different styles of play. These could then be used to change the experience of human opponents playing against them. They also investigate methods of improving the artificial agent’s strength, including parallelizing MCTS and using Association Rule Mining to predict opponent choices, thus improving their ability to play well against them.

Similarly to reward shaping, fitness based methods have shown promise in generating populations of diverse behaviours in a variety of domains [Stanley *et al.* 2005; Mitchell 2009; Hastings *et al.* 2009; Beukman *et al.* 2022]. Here various forms of behaviours emerged from designed fitness metrics. In a recent study Perez-Liebana *et al.* [2021] utilised a quality diversity algorithm, known as MAP-Elites, to produce a collection of agents that display a variety of playing styles. Nevertheless, as with reward-based techniques, a meticulous approach to the design of the behavioural descriptor is crucial. Additionally, the mapping of game-based features to specific behaviours is still ambiguous, rendering this design procedure non-intuitive.

3.4.2 Data-based

Imitation Learning (IL) is a commonly employed data-driven methodology for training a network to imitate a desired behaviour, be it from another algorithm or a human expert [Hussein *et al.* 2017]. During the imitation learning process, the algorithm observes the expert’s actions and tries to learn a mapping between the inputs (such as the state of the environment) and the corresponding outputs (such as the expert’s actions). This mapping can then be used to make decisions in similar situations. Imitation learning is often used in robotics [Lopes *et al.* 2007] and autonomous driving [Abbeel and Ng 2004], where it is difficult to program explicit rules and behaviours for the agents. By learning from expert demonstrations, the agents can acquire complex behaviours and skills that would be difficult to engineer manually. Silver *et al.* [2016] demonstrate the effectiveness of this approach by training an agent to play the game of Go at superhuman performance levels. This is achieved by utilising Imitation Learning as a pre-training step before utilising traditional RL. Q-learning from demonstrations is another bootstrapping approach which used demonstrations in Atari to bootstrap reinforcement learning against a known reward function rather than learning the reward function from the demonstrations [Hester *et al.* 2018]. A drawback to using IL is that the agent’s performance is limited to that of the demonstrator and in practice due to it being an approximation may even perform worse [Ross *et al.* 2011]. Additionally, depending on the scope and variety of the observations the entirety of the state space may be unexplored. If an agent finds itself in a region of uncertainty due to either error or the dynamic nature of the environment, it is unlikely for it to recover. This approach, therefore, requires a large number of expert examples to be effective. Obtaining these expert samples, however, can be difficult in terms of both time and cost, especially when working with human data. An Online Learning approach called DAGGER looked to mitigate the problem of overfitting by iteratively updating policies by requesting additional feedback from an expert [Ross *et al.* 2011]. This may be impractical for complex tasks with long training runs as the expert needs to be available throughout. Harmer *et al.* [2018] also acknowledged the difficulties of obtaining large amounts of expert data and instead applied imitation learning as a form of regulariser for a temporal difference RL agent. This allowed for improved performance and generalisability demonstrated in an in-house 3D game. Recently, impressive results in creating human-like agents have been demonstrated in the popular video game Counter-Strike GO² [Pearce and Zhu 2022]. Here, Pearce and Zhu [2022] utilised a supervised learning variant of IL called Behavioural Cloning (BC). Given our objective of training behaviours that showcase distinct styles, we intend to employ Imitation Learning (IL) in tandem with the play-style identification approach. This combination will enable us to effectively capture and replicate the nuanced behaviours associated with different play styles, enriching the training process and enhancing the fidelity of learned behaviours.

It has been shown that the combination of IL and RL alleviates some of the issues outlined. Examples of these are Approximate Policy Iteration with Demonstration (APID) and Inverse Reinforcement Learning

²<https://blog.counter-strike.net/>

(IRL). For APID, linear constraints are defined using the expert trajectories utilised by the optimisation process of a Policy Iteration algorithm [Kim *et al.* 2013]. The benefit of IRL approaches is the ability to learn a policy which can outperform the expert that it learnt from. Early IRL approaches assumed optimal demonstrations and a reward function that is linear in a known set of features. Here the key issue was that many differing reward functions could legitimately have led to a given optimal policy for some MDP. This led to Abbeel and Ng [2004] developing a method where both the recovered policy and expert obtain the same reward on the original MDP. This, however, did not allow for the policy to surpass the expert in performance. Since we look to learn policies which mimic observed behaviours, we do not require our policies to exceed the experts. Bayesian IRL embraced the idea of the existence of multiple valid reward functions by inferring a posterior distribution over rewards rather than committing to a single one [Ramachandran and Amir 2007]. By contrast, Maximum Entropy IRL returns a reward function that matches the expected feature counts, favouring rewards that lead to a higher-entropy stochastic policy [Ziebart *et al.* 2008]. These two techniques both allow for the learnt policy to outperform the expert-demonstrated policy. Ranchod *et al.* [2015] utilised both Bayesian and Maximum Entropy IRL to segment a set of unstructured demonstrations into a set of reusable “skills”. Specifically, in video games, there has been limited success in employing IRL. Uchibe [2018] looked to firstly train a classifier which could identify expert versus non-expert state transitions. This classifier was then used as a reward function to train a deep RL algorithm. However, their model rarely outperformed a BC baseline method. More recently an approach which looked to jointly learn the policy and reward function in an adversarial manner showed some promise in the Atari domain. However, they concluded that they still performed substantially worse than expert-level games warranting more work [Tucker *et al.* 2018].

3.5 Summary

Overall, this chapter serves as an exploration into how the related fields of advice generation, play-style identification, player modelling and imitation learning contribute to the development of effective automatic tailored advice generation systems. It also highlights the limitations of existing approaches and identifies opportunities for future research in this area. By doing so, this chapter provides a comprehensive overview of the state of the art in automatic tailored advice generation needed to guide our development of a novel solution to this complex problem.

Chapter 4

Research Methodology

Portions of this chapter have already been published [Ingram 2021]. *Branden Ingram. Generating tailored advice in video games through play-style identification and player modelling. In Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, volume 17, pages 228–231, 2021.*

Automated tailored advice generation in video games refers to the use of algorithms to provide customised advice to individual players based on their performance and gameplay patterns. The problem with this approach is that it requires a large amount of data about the player’s behaviour and preferences, as well as sophisticated machine learning algorithms to analyse that data and generate effective advice.

Additionally, the advice provided must be carefully crafted to avoid frustrating the player or detracting from the overall gaming experience. This can be challenging because players have different skill levels, play-styles, and goals, and what works well for one player may not work well for another. A system like this can result in increased player engagement and enhanced learning.

We aim to generate beneficial tailored advice for players through the identification of high-level play-styles and player modelling in video game domains. This approach involves acquiring a model of both the present state and the optimal expert version of an individual, specifically tailored to their play-style. It is through this method that we can draw comparisons and motivate the user to align their behaviour with their expert self. We suggest that by utilising this process, users can leverage the provided advice to enhance their performance and achieve higher levels of proficiency, specific to their play-style. To this end we propose a pipeline made up of four interconnected systems as outlined below:

- **The Play-style Identifier (PI):** Identifies, analyses and interprets all the different play-styles through unsupervised clustering (Chapter 5).
- **The Game Modeller (GM):** Learns a policy representative of the best way to play for each of the identified play-styles (Chapter 6).
- **The Player Modeller (PM):** Models the play-style of an individual user to be able to offer tailored advice (Chapter 7).
- **The Advisor (AD):** Compares a player’s model with their corresponding optimal play-style model to formulate useful advice (Chapter 8).

The concepts discussed in Chapter 2, as well as the knowledge gained from Chapter 3, were utilised to influence the design and aided with the implementation of these systems. The overall workflow of these individual components and how they interact with each other can be seen in Figure 4.1.

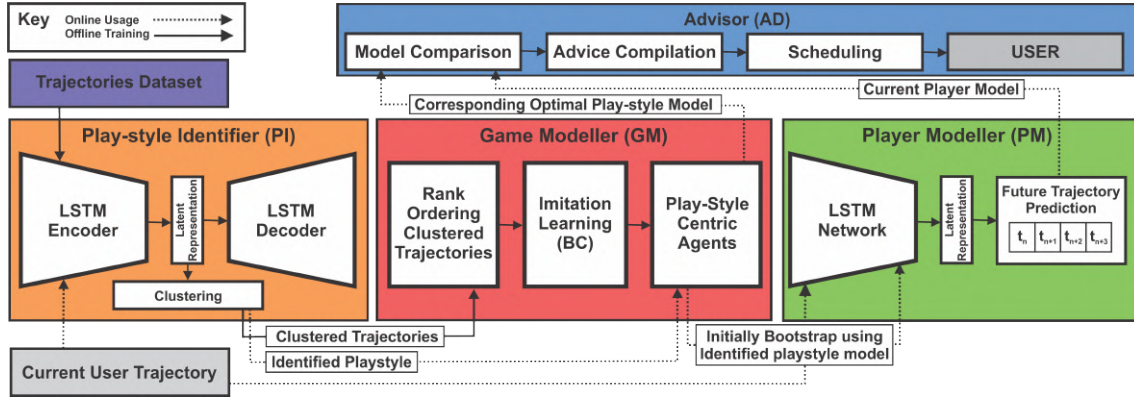


Figure 4.1: The complete Play-style-centric Advice Generation pipeline

Our methodology for generating customized advice revolves around the concept of comparing a model of the user’s gameplay style with a corresponding expert-level version of that same style. However, this requires a mechanism for understanding and interpreting the behaviour exhibited in the user’s gameplay. To accomplish this, we employ the use of the PI (orange block in Figure 4.1), which learns to identify various play-styles by compressing game trajectories and clustering them together. Subsequently, the GM (red block in Figure 4.1) utilises these groupings of play-styles to learn a set of play-style-centric policies that demonstrate a high level of proficiency corresponding to the characteristics of a particular play-style. Both the PI and GM are trained in an offline fashion using an existing dataset of gameplay trajectories. Through these offline processes, we can identify play-styles from trajectories as well as represent the optimal way to play for each play-style. Nonetheless, merely having this information is inadequate to offer personalised guidance, as it necessitates comprehending the behaviour of the individual receiving the advice. To formulate an understanding of a specific user, our PM (green block in Figure 4.1) uses an online training model which learns from live gameplay trajectories to predict future states and actions. This process is bootstrapped using both the PI and GM to initially improve prediction accuracy when limited user samples are available. Finally, both the expert model from the GM and the player model from the PM are utilised by AD (blue block in Figure 4.1) for advice compilation. Within this framework, the AD employs comparative analysis of the two models to determine areas of discrepancy and subsequently generates personalised recommendations aimed at motivating the user to adjust their behaviour in accordance with their respective expert.

The PI, GM, PM and AD components are all described in greater detail in Sections 4.1, 4.2, 4.3 and 4.4 respectively.

4.1 Play-style Identifier (PI)

The PI module as depicted in Figure 4.2 looks to answer the question: “**What are the different ways I can play a game?**”. This works by taking in input data in the form of simulated experiences or replay data, if available, for the domain being used. Specifically, this data consists of multidimensional varying-length time-series sequences known as gameplay trajectories. These trajectories encode the states visited by a player within an environment as well as the actions performed, during the process of playing a game. The function of the PI is, therefore, to be able to identify the set of different play-styles using this dataset of trajectories. This is achieved by using an autoencoder architecture to learn how to encode trajectories into a form which can be easily processed by clustering algorithms. In particular, we compress our trajectories into uniform length vectors which encode the behaviour of the original trajectory. The clustering process on these encodings, therefore, results in the separation of trajectories with respect to

play-style. Through the analysis of these identified clusters, we can determine characteristics associated with each play-style and their interrelations with one another, resulting in a more comprehensive and interpretable understanding. Thus, the PI is accountable for compressing complex trajectories into a form which can be clustered such that we can then identify the particular characteristics corresponding to all trajectories in a given dataset. The importance of this system lies not only in its capacity to identify individual user styles but also in its crucial role in acquiring policies that align with various play-styles. This component is discussed in greater detail in Chapter 5.

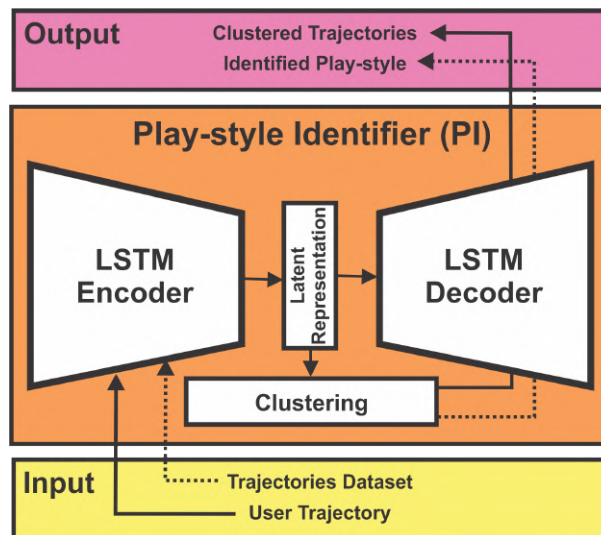


Figure 4.2: Overview of the Strategy Generator

4.2 Game Modeller (GM)

Knowing different ways to play a game does not guarantee improvement. Therefore, the GM module as depicted in Figure 4.3 looks to answer the question: “**What are the optimal ways to play in each style?**”. This process operates by producing a collection of agents that exhibit optimal performance within a defined domain, in accordance with a recognised behavioural pattern that resembles a particular play-style identified by the PI. These agents use a form of Imitation Learning on a subset of expert demonstrations separated through the usage of the PI. These expert demonstrations are obtained by selecting the top-performing trajectories based on a ranking metric. By training agents in an offline fashion on only expert demonstrations for each play-style, the GM can generate a set of optimal play-style-centric agents. Utilising this set of play-style-centric agents, our system can then be used to select the expert version of a user which corresponds to their play-style identified by the PI. The generation of these experts allows the AD to determine a behavioural direction towards which our users can head. Additionally, by selecting the averagely performing demonstrations, the GM can learn policies corresponding to the set of averagely performing play-style-centric agents. These weaker policies are used by the PM as a form of bootstrapping to improve performance. This component is discussed in greater detail in Chapter 6.

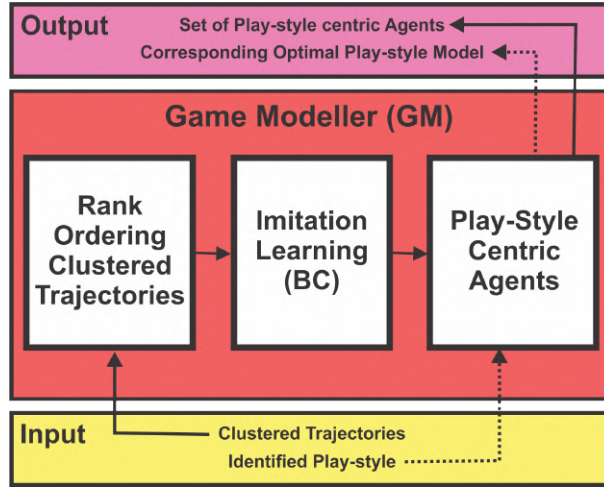


Figure 4.3: Overview of the Game Modeller

4.3 Player Modeller (PM)

While it is generally helpful to know the best ways to play in all identified play-styles, this alone is not enough to provide personalised advice without a specialised understanding of the individual in question. Therefore, the PM module as depicted in Figure 4.4 is utilised to answer the question: “**How does an individual user play?**”. To this end, we continuously train a supervised learning model in an online fashion on user trajectories to predict the future state and actions of an individual. To bootstrap this process, we use an averagely-performing play-style-centric model (learnt by the GM) that aligns with the play-style currently attributed to the user (identified by the PI). As the system gathers more information about the user’s play-style through additional gameplay, we can update the model accordingly to improve the model’s accuracy. This bootstrapping process is essential in mitigating the time gap between when the user starts playing and when our system produces meaningful tailored advice. This component is discussed in greater detail in Chapter 7.

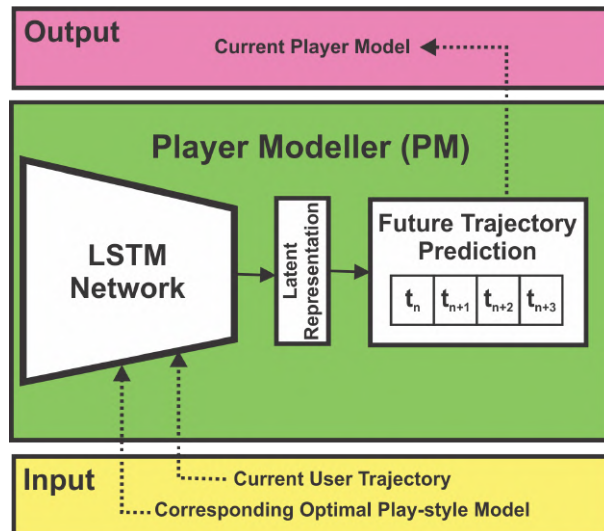


Figure 4.4: Overview of the Player Modeller

4.4 Advisor (AD)

The goal of the Advisor (AD) is to provide tailored advice by utilising the models learned by the Player Modeller (PM) and the Game Modeller (GM) to answer the question of “**what should be advised?**”. The GM represents the optimal version of an individual, while the PM represents the current behaviour of the user. The AD functions in three steps: model comparison, advice compilation, and advice scheduling, as illustrated in Figure 4.5. During the model comparison step, the low-level actions generated by the corresponding expert model (GM) are compared to the current player’s model (PM) to identify differences between the two. If any differences are found, a message is generated that indicates the user’s action and what the expert would have done, encouraging the user to act more in line with their corresponding expert. This message is then scheduled to be sent to the user based on a scheduling policy. The overall aim of the Advisor System is to reduce the differences between the two models, gradually guiding the user towards aligning with their particular expert style. Further details about this component are available in Chapter 8.



Figure 4.5: Overview of the Advisor

4.5 Domains

For the evaluation of the previously described advice generation model (Figure 4.1), we required a dataset of trajectories where the play-style of each was known. Although our model is unsupervised this condition needed to be satisfied to prove quantitatively the effectiveness of our approach. Obtaining a natural dataset of human trajectories of the appropriate size is not a trivial task within video games. This is due to the closed-source nature of commercial video games as well as the data legal issues around sharing user data. Additionally, having said dataset labelled specifically with respect to play-style is even rarer. For these reasons, we simulate data within a hand-designed environment. In particular, we required an environment which allowed for multiple, independent and easily observable strategies for solving a problem. For this purpose, we create our GridWorld environment described in Section 4.5.1 which is utilised to generate a dataset of trajectories using reward shaping. This environment serves as a testbed where quantifiable performances can be measured in terms of our play-style identification process. In addition, we use two additional environments to showcase the effectiveness of our pipeline in more complex settings spanning multiple levels, namely MiniDungeons (as described in Section 4.5.2) and a real-world environment with human-generated data, referred to as Mario (as detailed in Section 4.5.3).

4.5.1 GridWorld

To evaluate an algorithm for play-style identification, it is important to have multiple trajectories from a set of different play-styles. Trajectory-based datasets labelled according to style do not exist and therefore we generate data to account for this. We distinguish two play-styles as being different goals that could be reached by an agent. These we model as different reward functions in the reinforcement learning paradigm. This idea of reward shaping has been used to train a set of human-like agents with differing styles [Arzate Cruz and Ramirez Uresti 2018]. We utilise this approach to generate a set of trajectories with differing performance levels for multiple styles.

Our Preference-Based Trajectory Generation (PBTG) approach (Algorithm 4) was used to generate 5 individual datasets T_n from 5 different environments ($E_1, \dots, E_5$) with 4 varying play-styles (reward

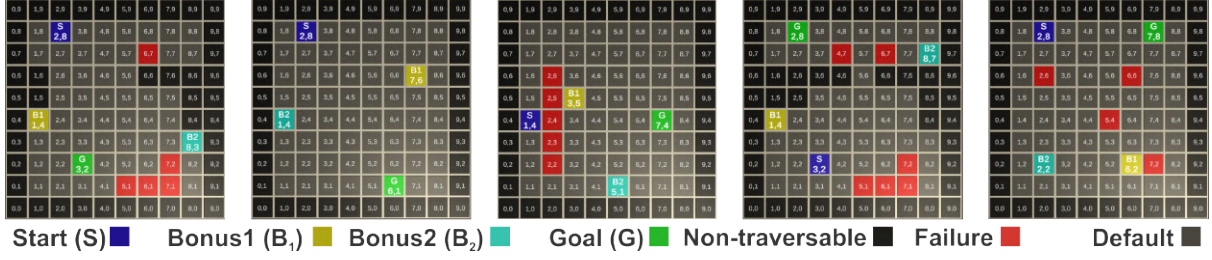


Figure 4.6: Randomly generated grid world environments $E_1, \dots, E_5$

functions) present. Each environment is a 10×10 grid world, as depicted in Figure 4.6. The environments each have a start state (S , in blue) and a goal state (G , in green). Walls (black tiles) cannot be traversed and trap states (red tiles) result in failure. The variety in play-styles is introduced through the addition of two bonus states (B_1 , in gold and B_2 , in cyan). These are the optional objectives that a player with certain preferences might wish to complete. The set of actions is the movement in any of the four primary cardinal directions. These environments were designed and implemented using Unity¹, a free-to-use 3D video game creation engine. These environments incorporated “ml-agents”², an open-source project that enables games and simulations to serve as environments for training intelligent agents. Lastly, these environments with the “ml-agents” toolkit were wrapped as an Open-AI gym³ environment using a pre-existing gym-wrapper⁴. The benefit of this approach is the ability to utilise the flexibility of Unity for design and the power of standard-baseline⁵ implementations of reinforcement learning algorithms compatible with OpenAI gym.

Using this approach we generated data with 4 play-styles, as described in Table 4.1. These are the observable behaviours our model aims to recover through its play-style identification process. The set of reward functions R used to emulate these behaviours is also defined in Table 4.1. Here we defined large positive rewards for the objectives we wished the agent to accomplish. The respective bonus rewards were only given the first time an agent reached either B_1 or B_2 . Specifically, the actions of R_1 entail the desire of the agent to travel in a direct path towards the goal. On the other hand, R_2 aims to reach the goal through the intermediary point of B_1 , while R_3 seeks to travel via B_2 . Finally, R_4 intends to navigate to the goal by passing through both B_1 and B_2 . Following the procedure outlined in Algorithm 4 on the following page we trained a Q-learning agent for each of the combinations of R and E for 20000 episodes with discount factor $\gamma = 0.99$ and linear ϵ -decay over the number of training episodes to ensure our agent converges to the global optimum. The state is given by the tuple (x, y, b_1, b_2) where x and y are the Cartesian grid coordinates and b_1 and b_2 indicate whether an agent has visited B_1 or B_2 , respectively. Our dataset consists of 8000 randomly selected trajectories per R and E . Therefore our trajectory is a sequence of time steps in the form of (x, y, b_1, b_2) .

¹<https://unity.com/>

²<https://github.com/Unity-Technologies/ml-agents>

³<https://github.com/openai/gym>

⁴<https://github.com/gzrjzcx/ML-agents/blob/master/gym-unity/README.md>

⁵<https://stable-baselines3.readthedocs.io/en/master/>

Table 4.1: Observable play-styles and reward structure

R	Behaviour	G reward	B_1 reward	B_2 reward
1	Moves directly to G	100	0	0
2	Visits B_1 before G	100	50	0
3	Visits B_2 before G	100	0	50
4	Visits B_1 and B_2 before G	100	50	50

Algorithm 4 Preference-Based Trajectory Generation (PBTG)

```

1: procedure PBTG(Environment  $E$ )
2:   Define a set of reward functions  $R$ 
3:   Initialise our set of trajectories  $\mathcal{D} = \{\}$ 
4:   for all reward functions  $r \in R$  do
5:     Use Q-learning to learn optimal policy  $\pi_r^*$ 
6:     for  $n$  number of required Trajectories do
7:        $\pi_r^n \leftarrow \text{perturb}(\pi_r^*)$ 
8:       Generate  $X(n, r)$  from  $\pi_r^n$  and append to  $\mathcal{D}$ 

```

Appendix A.1 provides heatmap visualizations for all GridWorld environments, as well as separate heatmaps for trajectories categorized by reward function. This feature allows for a visual depiction of the encoded behaviour for each of the four play-styles. Table 4.2 below depicts the diversity statistics across the trajectories generated using Algorithm 4. Here the diversity is calculated by firstly padding trajectories such that they are the same length. Once both trajectories are the same size, we calculate the difference between them by taking the Root Mean Square Error (RMSE), which measures how different they are.

Table 4.2: Diversity analysis of GridWorld environments including average difference calculated using RMSE on padded trajectories

	E_1	E_2	E_3	E_4	E_5
Number of times agent reached the goal	6399	4783	8000	5997	5775
Number of times agent failed	1600	3216	0	2002	2224
Average trajectory length	23.05	25.05	24.88	14.87	22.06
Variance in trajectory length	197.98	127.56	194.85	58.25	187.49
Min trajectory length	7	3	11	1	2
Max trajectory length	119	106	115	50	136
Average trajectory diversity	2.17	2.68	2.35	2.31	2.51
Variance in trajectory diversity	0.41	0.56	0.49	0.51	0.38
Min trajectory diversity	0	0	0	0	0
Max trajectory diversity	3.84	4.58	3.91	4.14	4.38

In summary, we generated a dataset of varied noisy trajectories across multiple different levels whose behaviour corresponded with one of the four behaviours described in Table 4.1. If we consider, for example, E_4 we observe in Figure 4.7 the general behaviour for each play-style represented as a sequence of notable state transitions. However, it's important to clarify that these depicted state transitions are not an exhaustive representation of every possible transition in the system, but rather a concise sum-

mary highlighting the critical checkpoints that characterise the behaviour of each play-style. It is these behaviours which we looked to recover in this test-bed domain.

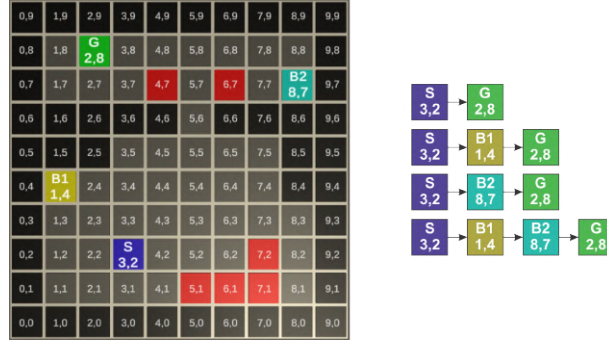


Figure 4.7: General behaviour (sequence of notable state transitions) of each of the four play-styles corresponding to E_4

4.5.2 MiniDungeons

MiniDungeons is a two-dimensional top-down dungeon exploration game, and is a common benchmark research domain for modelling and understanding human play-styles [Holmgard *et al.* 2014]. We use a dataset of player trajectories generated using six player proxies which are handcrafted policies [Arendse *et al.* 2022]. Table 4.3 describes these different play-styles corresponding to the six player proxies as well as their corresponding behaviour with example trajectories depicted in Figure 4.8. The state at each timestep is defined as a 15 dimensional vector which encodes event information up until that current timestep and aims to track higher-level player tactics. We combine trajectories generated across multiple different levels to form our training and testing sets. The specific levels used for each of these sets, as originally designated by Holmgard *et al.* [2014], are listed as follows:

- *train* (md-test-v0, md-hard-v0, md-random_1-v0, md-random_2-v0, md-gene_1-v0, md-gene_2-v0, md-strand_1-v0)
- *test* (md-strand_2-v0, md-holmgard_0-v0, md-holmgard_3-v0, mmd-holmgard_5-v0, md-holmgard_7-v0, md-holmgard_8-v0, md-holmgard_1-v0, md-holmgard_2-v0, md-holmgard_4-v0, md-holmgard_6-v0, md-holmgard_9-v0, md-holmgard_10-v0)

These levels are all visualised in Appendix A.2.

Table 4.3: MiniDungeons player proxy behaviours

Play-style	Behaviour
Safe Runner	Complete the dungeon as quickly as possible while avoiding monsters
Wreckless Runner	Complete the dungeon as quickly as possible without avoiding monsters
Brave Treasure Hunter	Collect treasure even if guarded by enemies
Pure Treasure Hunter	Collect only unguarded treasure
Safety-First	Only collect treasure and potions
Monster Killer	Fight as many monsters as possible without dying

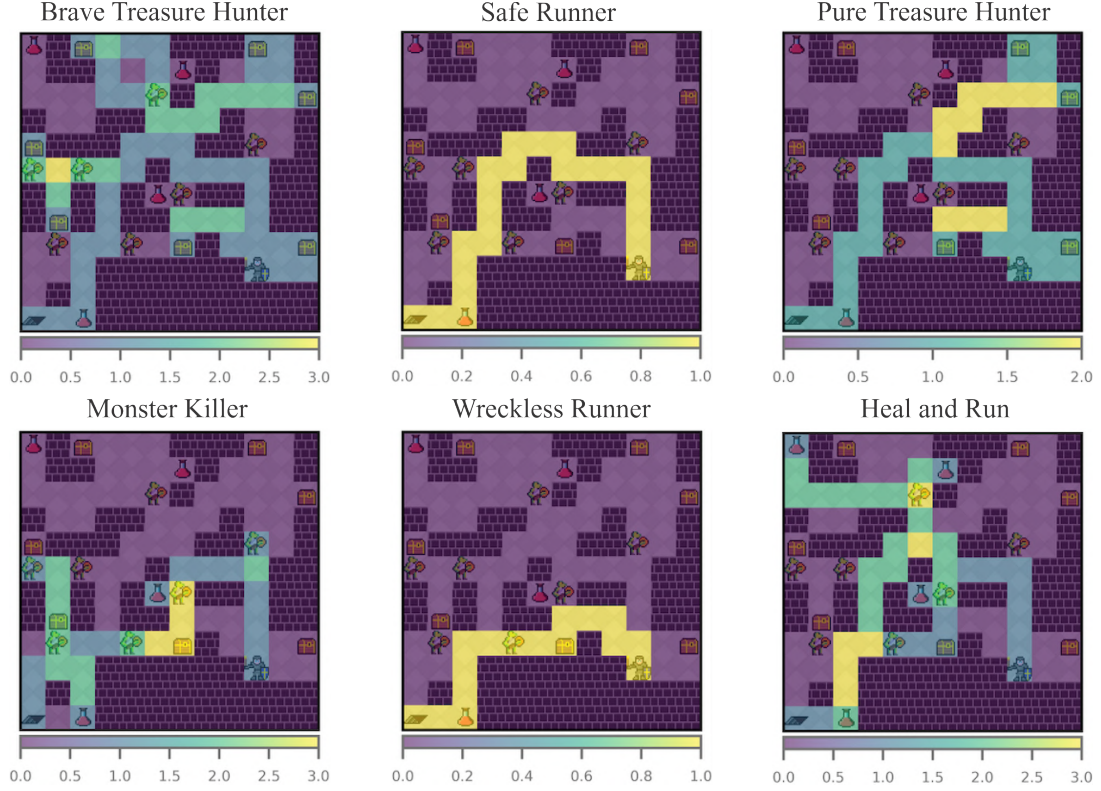


Figure 4.8: Example visualisation for each of the 6 player proxies in the MiniDungeons domain [Holmgard et al. 2014]

An example of three trajectories can be seen in Figure 4.9, where each 15-dimensional time-step is separated by the “|” character.

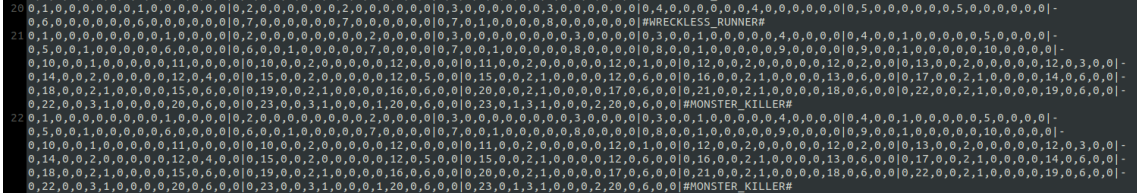


Figure 4.9: Example of 3 trajectories extracted from the MiniDungeons dataset

4.5.3 Mario

This dataset consists of 74 playthroughs across 11 different levels of Super Mario Bros. An example level from this set is visualised in Figure 4.10. These playthroughs were each captured by logging the actions of a unique human participant [Guzdial and Riedel 2016]. We then refactored this data into a trajectory, where each time step represents the current state of the playthrough at that point. We defined a state as a tuple given by (j, k, r, c, d, e) where j is the number of jumps, k is the number of enemies killed, r number of times the player has started running, c the number of coins collected, d the number of times the player died and e the unique encoding for each action.



Figure 4.10: Example visualisation of a level in the Mario domain

The complete set of levels used for capturing trajectories can be found in Appendix A.3. An example of trajectories can be seen in Figure 4.11, where each 6-dimensional time-step is separated by the “|” character.

```
0,0,0,0,0,0|0,0,0,0,0,1|0,0,0,0,2|1,0,0,0,3|1,0,0,0,4|1,0,0,0,5|2,0,0,0,3|2,0,0,0,4|2,0,0,0,6|2,0,0,0,7|2,0,0,0,8|2,0,0,0,9|2,0,0,0,2|
3,0,0,0,3|3,0,0,0,4|4,0,0,0,3|4,0,0,0,4|5,0,0,0,3|5,0,0,0,2|5,0,0,0,4|6,0,0,0,3|6,0,0,0,2|6,0,0,0,2|6,0,0,0,2|6,0,0,0,9|
6,0,0,0,2|6,0,0,0,9|6,0,0,0,2|6,0,0,0,4|6,0,0,0,9|7,0,0,0,3|7,0,0,0,4|8,0,0,0,3|8,0,0,0,2|8,0,0,0,9|8,0,0,0,2|8,0,0,0,2|8,0,0,0,9|
8,0,0,0,2|8,0,0,0,10|8,0,0,0,9|8,0,0,0,2|8,0,0,0,9|8,0,0,0,4|8,0,0,0,9|8,0,0,0,9|9,0,0,0,3|9,0,0,0,4|10,0,0,0,3|10,0,0,0,11|10,0,0,0,0|
10,1,0,0,12|10,1,0,0,9|10,1,0,0,4|10,1,0,0,6|10,1,0,0,9|10,1,0,0,5|10,1,0,0,2|11,1,0,0,3|11,1,0,0,4|11,1,0,0,8|11,1,0,0,6|11,1,0,0,2|
12,1,0,0,3|12,1,0,0,4|13,1,0,0,3|13,1,0,0,4|14,1,0,0,3|14,1,0,0,2|14,1,0,0,2|14,1,0,0,4|15,1,0,0,3|15,1,0,0,9|15,1,0,0,10|15,1,0,0,4|
15,1,0,0,2|15,1,0,0,9|16,1,0,0,3|16,1,0,0,9|16,1,0,0,4|17,1,0,0,3|17,1,0,0,9|17,1,0,0,2|17,1,0,0,4|18,1,0,0,3|18,1,0,0,10|18,1,0,0,4|
18,1,0,0,9|19,1,0,0,3|19,1,0,0,4|20,1,0,0,3|20,1,0,0,4|21,1,0,0,3|21,1,0,0,4|21,1,0,0,2|21,1,0,0,2|22,1,0,0,3|22,1,0,0,2|22,1,0,0,9|
22,1,0,0,2|22,1,0,0,9|22,1,0,0,9|22,1,0,0,4|22,1,0,0,2|22,1,0,0,2|22,1,0,0,9|22,1,0,0,2|23,1,0,0,3|23,1,0,0,2|23,1,0,0,9|23,1,0,0,9|
23,1,0,0,9|23,1,0,0,2|23,1,0,0,4|23,1,0,0,9|24,1,0,0,3|24,1,0,0,4|25,1,0,0,3|25,1,0,0,4|26,1,0,0,3|26,1,0,0,9|26,1,0,0,4|27,1,0,0,3|
27,1,0,0,4|28,1,0,0,3|28,1,0,0,4|28,1,0,0,2|29,1,0,0,3|29,1,0,0,2|29,1,0,0,2|29,1,0,0,9|29,1,0,0,9|29,1,0,0,4|29,1,0,0,2|29,1,0,0,9|
29,1,0,0,2|30,1,0,0,3|30,1,0,0,2|30,1,0,0,9|30,1,0,0,2|30,1,0,0,9|30,1,0,0,3|31,1,0,0,3|31,1,0,0,4|32,1,0,0,3|32,1,0,0,9|
32,1,0,0,4|32,1,0,0,9|32,1,0,0,7|32,1,0,0,2|32,1,0,0,11|32,1,0,0,9|32,1,0,0,13|
```

Figure 4.11: Example of a single Mario domain trajectory

4.6 Summary

Within this chapter, we have provided a structured framework of our comprehensive model, encompassing its functionality, along with a formal introduction to its constituent components. Furthermore, we have presented the domains in which we aim to showcase the efficacy of our model. The ensuing chapters describe each of the aforementioned components in detail.

Chapter 5

Play-style Identification

Portions of this chapter have been published in [Ingram et al. \[2022a\]](#), with further extensions published in [Ingram et al. \[2023b\]](#).

Branden Ingram, Benjamin Rosman, Richard Klein, and Clint van Alten. Play-style identification through deep unsupervised clustering of trajectories. In 2022 IEEE Conference on Games (CoG). IEEE, 2022.

Branden Ingram, Benjamin Rosman, Richard Klein, and Clint van Alten. Generating Interpretable Play-style Descriptions through Deep Unsupervised Clustering of Trajectories. In 2023 IEEE Transactions on Games (ToG). IEEE, 2023.

5.1 Introduction

Play-style identification is an important concept in various fields, including sports, video games, music, and even business. Identifying a player’s play-style can help coaches, teammates, or managers to understand their strengths and weaknesses, and tailor their training or strategies accordingly [[Charles et al. 2005](#)]. This can lead to improved performance and better results. In particular, for video game developers tailoring user experience has become an important goal required to create more engaging and personalised experiences for players. By understanding how different players like to play, game designers can create content and features that cater to their preferences, making the game more enjoyable.

Identifying play-styles can be challenging for several reasons. Firstly, play-styles can be highly subjective and can vary significantly between different players, making it difficult to create a single, universally applicable model. Additionally, play-styles can be influenced by a wide range of factors, including individual preferences, skill levels, experience, and personality traits. Furthermore, play-styles can be difficult to identify accurately, particularly when only partial or incomplete information is available. Players may exhibit different play-styles in different situations, making it challenging to create a comprehensive and reliable model.

Numerous studies have examined modelling an individual’s problem-solving style, which is not exclusive to games. For instance, in the realm of education, researchers have attempted to identify learning styles [[Dunn 1984](#)]. Similarly, in gaming, investigations have aimed to identify player archetypes, personality traits, and motivators [[Bartle 1996](#); [Yee 2005](#); [Drachen et al. 2009](#)]. Although work has been done to recognise play-style, little effort has been devoted to employing these techniques to analyse complex trajectory-based data. Without considering the temporal aspect of a player’s gameplay, we risk overlooking the sequence of events that led to a particular state and instead solely focus on the final

state. This approach may obscure valuable insights that could otherwise be obtained by using detailed features. Furthermore, as video games are time-series domains, where decision-making varies throughout gameplay, it is appropriate to model play-styles using temporal models.

Moreover, identifying play-style from large datasets and interpreting generalised behaviours remains challenging. Non-trajectory-based data (summary statistics) have been utilised in video games to characterise player roles [Eggert *et al.* 2015]. Developing such a technique for trajectory data requires compressing both the spatial and temporal distributions. Recent studies have employed clustering techniques to produce such descriptions [Xie *et al.* 2016; Drachen *et al.* 2014]. These descriptions offer valuable insights into the features and patterns present in the underlying data; however, they have yet to be employed in video game domains as informative tools for both players and designers.

In this chapter, we address both the problem of processing complex data and that of finding interpretive descriptions of behaviours within a dataset. These descriptions look to describe some general characteristics or patterns exhibited by a group, useful for tailoring user experiences. To that end, we demonstrate a novel system for play-style identification through the clustering of multi-dimensional variable-length trajectories in video games and demonstrate that these clusters represent varying styles of behaviours. This system and its functioning are formally defined in Section 5.2. The core of this system is a specialised LSTM-autoencoder that utilises the benefits of Recurrent Neural Network [Hochreiter and Schmidhuber 1997] architectures to handle sequence-based data. We evaluate this model on a generated benchmark dataset as well as in a natural domain (Mario). In Sections 5.4.1 and 5.4.2, we showcase the capability of our model to discern the style utilising incomplete (partial) trajectories, without necessitating supplementary data engineering or training time, in both offline and online contexts. This suggests the model is highly efficient and robust enough to be utilised for video games. In Section 5.4.3 we utilise this model to recover the characteristics of each play-style through state-based cluster analysis. Specifically, the similar, differing and unique states across clusters are identified. Furthermore, we demonstrate our approach to identifying the decision boundaries that separate different play-styles in Section 5.4.4. Finally, in Sections 5.4.5 and 5.4.6, we demonstrate a process for generating mean representations of each play-style as well as for determining the appropriate number of play-styles. These analytic processes were used to further describe the relational characteristics between the identified cluster.

5.2 Methodology

We aim to solve the following problem: Given a set of trajectories ($\mathcal{D}$) can we learn to separate them into k distinct partitions ($\mathcal{P}_k \subseteq \mathcal{D}$) with respect to unknown play-style characteristics. Can these partitions then be used to identify interpretable characteristics describing each play-style as well as their relationships?

To this end, we developed a fully unsupervised clustering approach for the identification and analysis of play-styles as depicted in Figure 5.1. Our unsupervised approach is based on three key steps. First, we utilise an LSTM autoencoder network to project a trajectory into a lower-dimensional latent representation. We then perform clustering on this latent space to discover clusters corresponding to related trajectories in this space. Lastly, we perform multiple forms of cluster analysis to obtain the cluster characteristics, decision boundaries and general behavioural representations for each cluster.

5.2.1 Trajectory Encoding

An autoencoder works to reconstruct each original input trajectory ($X_i \in \mathcal{D}$) after first encoding it as a lower-dimensional state ($Z_i \in \mathcal{Z}$). Formally, this is given by Equations 5.1 and 5.2.

$$Z_i = \text{Encoder}(X_i) \tag{5.1}$$

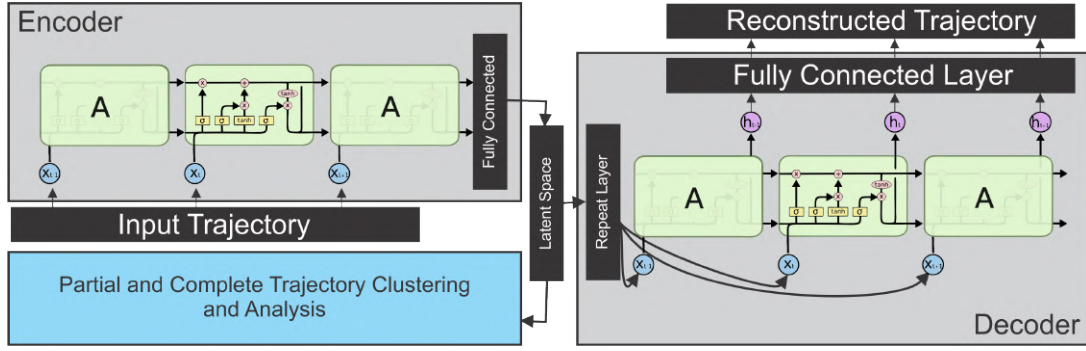


Figure 5.1: Play-style Identifier - LSTM autoencoder and clustering model

$$X'_i = \text{Decoder}(Z_i) \quad (5.2)$$

Our specific model is a temporal autoencoder containing non-stacked LSTM layers similar to [Xie *et al.* \[2016\]](#). This allows the processing of varied length trajectories by feeding the state at each time step into its own LSTM cell. These cells (“A” in Figure 5.1), learn to remember important information in sequence until finally outputting a fixed-sized vector representative of our latent space ($\mathcal{Z}$) by applying Equation 5.1. The latent representation is then decoded using Equation 5.2 to obtain X'_i , the reconstructed trajectory. The network is trained using back-propagation through time [[Hochreiter and Schmidhuber 1997](#)].

5.2.2 Trajectory Clustering

Having projected the trajectories into the latent space, we then cluster them. This clustering step is performed on the set of all generated pairs of (X_i, Z_i) , where $X_i \in \mathcal{D}$ and Z_i is the output of the encoder in Equation 5.1. Each pair (X_i, Z_i) is clustered with respect to Z_i to form predicted labels y'_i which are the cluster identifiers. Since Z_i is a representation of X_i we can use y'_i as the cluster label for the original data. Clustering using $\mathcal{Z}$ enables the use of standard clustering algorithms as there is no longer an issue of varying length or temporal features. This particular aspect of our model is depicted in Figure 5.2. Through the clustering process, we also obtain the set of partitions ($\mathcal{P}$) with associated centroids ($\mathcal{C}$) where each partition $\mathcal{P}_k$ stores the associated trajectories based on the predicted label where $k = y'$.

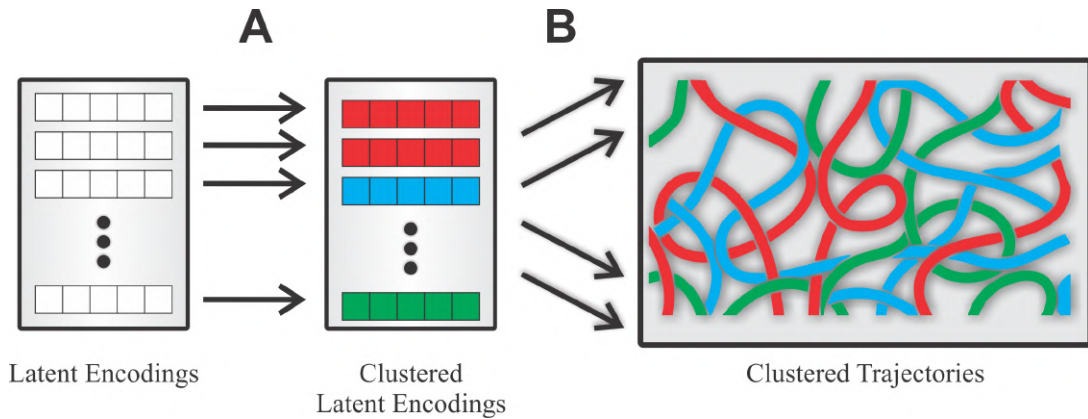


Figure 5.2: Play-style Identifier - Latent space clustering of trajectories

5.2.3 Cluster Analysis

After clustering all pairs of (X_i, Z_i) to obtain a tuple $(X_i, Z_i, y'_i) \forall i$ we conduct four forms of analysis as outlined below:

- **Offline and Online Clustering Performance Analysis:** A validation process to demonstrate clustering performance (Section 5.2.3.1)
- **Play-style Characteristic Identification:** The process of identifying the shared, differing, and unique states among different playstyles and in relation to them (Section 5.2.3.2).
- **Decision Boundary Identification:** The process of identifying the states where one play-style crosses over with another (Section 5.2.3.3).
- **General Behaviour Identification:** The process of determining a “mean” play-style example for each play-style (Section 5.2.3.4).

These processes are crucial for creating interpretable descriptions that formalise the characteristics and dynamics of each cluster, and provide a comprehensive and insightful view of the underlying data.

5.2.3.1 Offline and Online Clustering Performance

Although our model is completely unsupervised, we do compare the ground truth cluster labels (y_i) with the predicted labels (y'_i) to validate accuracy on complete trajectories. This validation step uses Equations 5.3, 5.5 and 5.6. Firstly Equation 5.3 finds the best match between the cluster assignments from an unsupervised algorithm (y'_i) and a ground truth assignment (y_i), M ranges across all possible one-to-one mappings and N is the number of data points [Kuhn 1955].

$$\text{Map} = \max_M \left(\frac{\sum_{i=1}^N 1\{y_i = M(y'_i)\}}{N} \right) \quad (5.3)$$

An example output mapping for four clusters can be seen in Figure 5.3. It is necessary to find the “best possible mapping” between the predicted and ground truth cluster assignments because the predicted labels may not always correspond exactly to the ground truth labels. The clustering algorithm may assign different data points to different clusters than the ground truth labels, which can result in a different clustering structure. Thus, a direct comparison between the predicted and ground truth labels may not accurately reflect the algorithm’s true performance.

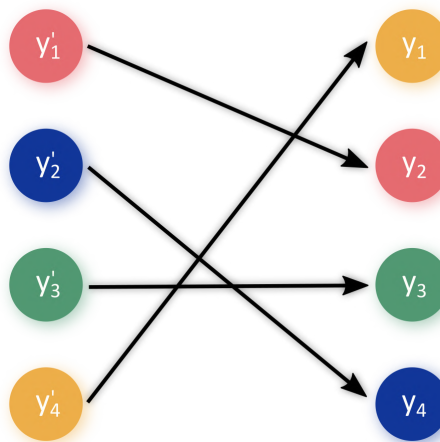


Figure 5.3: Example mapping between predicted labels y'_i and ground truth labels y_i

Secondly, Equation 5.5 defines a confidence measure ($\bar{k}$) giving the probability vector that a given trajectory X belongs to a particular cluster. This is determined using Equation 5.4 which calculates the Euclidean distance vector $\bar{d}$ between X and $\mathcal{C}$, where $\mathcal{C}$ is the set of centroids determined through the clustering step. The fundamental idea is to determine the distance between the encoded data point X and each centroid in the set $\mathcal{C}$. As a centroid ($\mathcal{C}_k$) approaches X in proximity, the likelihood that it is a member of cluster k increases correspondingly.

$$\bar{d}(X) = \|\mathcal{C} - \text{Encoder}(X)\|_2^2 \quad (5.4)$$

$$\bar{k}(X) = \frac{\exp(\bar{d}^{-1}(X))}{\sum \exp(\bar{d}^{-1}(X))} \quad (5.5)$$

Finally, we use Equation 5.6 along with the mapping learned in Equation 5.3 and the confidence $\bar{k}$ to determine if the prediction ($\text{Map}(\text{argmax}(\bar{k}))$) equals the ground truth y . The method functions by initially detecting the cluster assignment with the highest probability ($\text{argmax}(\bar{k})$), then associating it with the most probable label via Equation 5.6. Subsequently, this label is compared to the actual ground truth value y .

$$\text{Correct_Prediction}(X, y) = \begin{cases} 1 & \text{if } y = \text{Map}(\text{argmax}(\bar{k})) \\ 0 & \text{if } y \neq \text{Map}(\text{argmax}(\bar{k})) \end{cases} \quad (5.6)$$

Additionally, since identifying play-styles on partial trajectories is a key feature, we need to perform a similar evaluation step on these partial trajectories. To this end, we calculate the total accuracy using Algorithm 5 as the average correctly labelled predictions for every trajectory ($X \in \mathcal{D}$). In this case, our predicted label (y') is determined by first calculating the weighted moving average confidence ($\overline{\text{WMAC}}$) [Zhuang et al. 2007] over all partial trajectories using Equation 5.7. Here $\bar{k}_i$ represents the confidence calculated using Equations 5.4 and 5.5 as $\bar{k}(P)$ where $P = X[0 : i]$ and the weighting $w \leftarrow \frac{m(m+1)}{2}$ where m is the length of X . The cluster with the highest confidence is then selected as our predicted label and compared to the corresponding ground truth label y . This process is then repeated for all $X \in \mathcal{D}$ with our final partial trajectory accuracy (PTA) being the percentage of correct predictions.

$$\overline{\text{WMAC}} = \sum_{i=0}^m \left[\frac{\bar{k}_i \times i}{w} \right] \quad (5.7)$$

Algorithm 5 Partial Trajectory Accuracy (PTA)

```

1: procedure PTA( $\mathcal{D}$ )
2:    $PTA = 0$ 
3:   for all  $X \in \mathcal{D}$  do
4:      $m \leftarrow \|\mathbf{X}\|$   $\triangleright m$  is the length of  $X$ 
5:      $w \leftarrow \frac{(m)(m+1)}{2}$   $\triangleright w$  is a weighting over values of  $m$ 
6:     Calculate  $\overline{\text{WMAC}}$  for all partial trajectories using (5.7)
7:      $y' \leftarrow \text{argmax}(\overline{\text{WMAC}})$   $\triangleright y'$  is the predicted cluster
8:     if  $y'$  matches ground truth  $y$  then
9:        $PTA \leftarrow PTA + 1$ 
10:   $PTA \leftarrow \frac{PTA}{n}$   $\triangleright n$  is the number of trajectories in  $\mathcal{D}$ 

```

5.2.3.2 Identifying Play-style Characteristics

As clustering is an unsupervised approach it is not commonly the case that the semantic meaning behind clusters is known beforehand. The characteristics of each cluster need to be identified for the grouping to be useful for developers or players. For the purpose of identifying play-style characteristics we utilise state-based frequency analysis. In particular we use Equation 5.8 to calculate the frequency (f_k) of occurrences for a particular state (σ). In Equation 5.8 a state ($s_{t,i}$) is defined as an individual time step (t) within an i -th trajectory from $\mathcal{P}_k$ of length m_t . Here, the state $\sigma \in \mathcal{S}$ and $\mathcal{S}$ is the set of all possible states in the partition $\mathcal{P}_k$. Here we also denote n_k to be the total number of trajectories for a given play-style partition $\mathcal{P}_k$.

$$f_k(\sigma) = \sum_{t=1}^{n_k} \left[\sum_{i=1}^{m_t} 1[s_{t,i} = \sigma] \right] \quad (5.8)$$

The functioning of Equation 5.8 is demonstrated in Figures 5.4 and 5.5 where we utilise a simple example. Given a sequence of states making up two trajectories for each play-style, we obtain the formulation depicted in Figure 5.4 where $\mathcal{S}$ now becomes the set of all states within these trajectories.

$$\begin{aligned} f_{k_1} & \left(\begin{array}{l} t=[2,8,1,1|3,8,1,1|3,7,1,1|3,6,1,1|4,6,1,1|3,6,1,1|3,5,1,1|3,4,1,1|3,3,1,1] \\ t=[2,8,1,1|2,7,1,1|3,7,1,1|3,6,1,1|3,5,1,1|3,4,1,1|3,3,1,1|4,3,1,1|5,3,1,1|5,2,1,1|4,2,1,1|4,1,1,1] \end{array} \right) \\ f_{k_2} & \left(\begin{array}{l} t=[2,8,1,1|2,7,1,1|2,6,1,1|2,5,1,1|2,4,1,1|1,4,0,1|2,4,0,1|3,4,0,1|3,3,0,1] \\ t=[2,8,1,1|2,7,1,1|2,6,1,1|2,5,1,1|3,5,1,1|2,5,1,1|3,5,1,1|4,5,1,1|3,5,1,1|2,5,1,1|2,4,1,1|1,4,0,1|2,4,0,1|2,3,0,1|2,2,0,1] \end{array} \right) \end{aligned}$$

Figure 5.4: Example formulation of f_k given a set of specific trajectories where $k = 1, 2$

By applying Equation 5.8 to the example in Figure 5.4 we obtain the resulting state frequencies depicted in Figure 5.5. For example, given state (2, 7, 1, 1) has a frequency of 1 and 2 for f_{k_1} and f_{k_2} respectively.

$$\begin{aligned} f_{k_1} &= \left\{ [(2,7,1,1):1], [(2,8,1,1):2], [(3,3,1,1):2], [(3,4,1,1):2], [(3,5,1,1):2], [(3,6,1,1):3], [(3,7,1,1):1], \right. \\ & \quad \left. [(3,8,1,1):1], [(4,1,1,1):1], [(4,2,1,1):1], [(4,3,1,1):1], [(4,6,1,1):1], [(5,2,1,1):1], [(5,3,1,1):1] \right\} \\ f_{k_2} &= \left\{ [(2,4,1,1):2], [(2,5,1,1):2], [(2,6,1,1):2], [(2,7,1,1):2], [(2,8,1,1):2], [(3,5,1,1):2], [(4,5,1,1):1], \right. \\ & \quad \left. [(1,4,0,1):2], [(2,2,0,1):1], [(2,3,0,1):1], [(2,4,0,1):2], [(3,3,0,1):1], [(3,4,0,1):1] \right\} \end{aligned}$$

Figure 5.5: Example of neighbouring states for two states namely (5, 6) and (4, 2)

Therefore the application of Equation 5.8 on each play-style allows us to form an interpretation of general characteristics associated with that play-style. Although the identification of individual cluster characteristics has been previously demonstrated [Drachen *et al.* 2009], very little research has been done in terms of the relationship between play-styles in video games. To identify play-style characteristics in relation to each other we apply Equation 5.8 to identify the shared, differing and unique states across all play-styles as depicted in Figure 5.6.

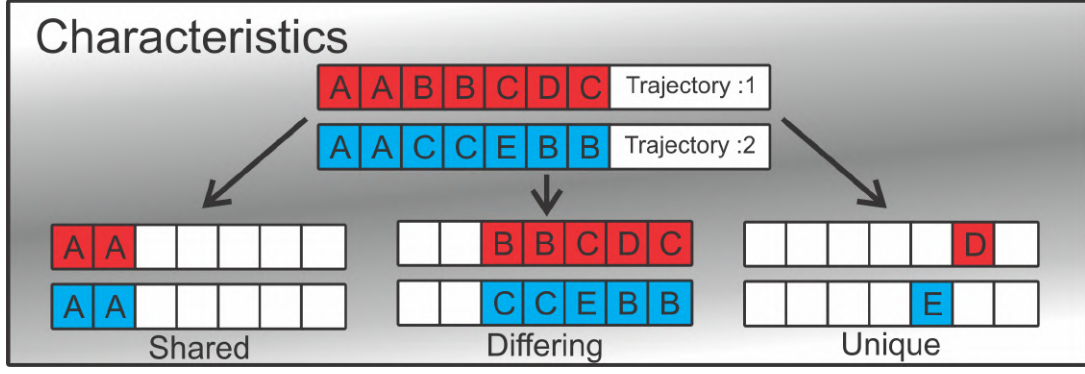


Figure 5.6: Example representation of the play-style characteristic identification process

Shared Characteristics

The shared states between play-styles are calculated as the combined frequency for each common state. A state is said to be common if it exists within trajectories in both clusters. Frequencies are combined by selecting the minimum frequency between the two clusters for each particular state (σ). Formally this is defined in Equation 5.9 where k_1, k_2 correspond to the 2 clusters whose shared states are being calculated.

$$Shared(\sigma, k_1, k_2) = \min(f_{k_1}(\sigma), f_{k_2}(\sigma)) \quad (5.9)$$

Intuitively, here we are obtaining the regions where the state visitations overlap, which can be likened to the intersection operator from boolean algebra [Sikorski and others 1969]. Once again we utilise the simple example as depicted in Figures 5.4 and 5.5 to demonstrate the functioning of Equation 5.9. By applying Equation 5.9 to this example we obtain the results depicted in Figure 5.7. Here it is observed that the only states in common are (2, 7, 1, 1), (2, 8, 1, 1) and (3, 5, 1, 1) (depicted in orange) and after applying the “min” operation, these are the only states which remain. If we consider state (3, 3, 1, 1) which has a frequency of 2 and 0 in f_{k_1} and f_{k_2} respectively, since 0 is the minimum value it is not included in the final set of shared states.

$$f_{k_1} = \left\{ \begin{array}{l} [(2,7,1,1):1], [(2,8,1,1):2], [(3,3,1,1):2], [(3,4,1,1):2], [(3,5,1,1):2], [(3,6,1,1):3], [(3,7,1,1):1], \\ [(3,8,1,1):1], [(4,1,1,1):1], [(4,2,1,1):1], [(4,3,1,1):1], [(4,6,1,1):1], [(5,2,1,1):1], [(5,3,1,1):1] \end{array} \right\}$$

$$f_{k_2} = \left\{ \begin{array}{l} [(2,4,1,1):2], [(2,5,1,1):2], [(2,6,1,1):2], [(2,7,1,1):2], [(2,8,1,1):2], [(3,5,1,1):2], [(4,5,1,1):1], \\ [(1,4,0,1):2], [(2,2,0,1):1], [(2,3,0,1):1], [(2,4,0,1):2], [(3,3,0,1):1], [(3,4,0,1):1] \end{array} \right\}$$

$$\min(f_{k_1}, f_{k_2}) = \left\{ [(2,7,1,1):1], [(2,8,1,1):2], [(3,5,1,1):2] \right\}$$

Figure 5.7: Shared states observed between f_{k_1} and f_{k_2} in the simple example

Differing Characteristics

The difference between two play-styles at a state-based level is calculated using Equation 5.10.

$$Diff(\sigma, k_1, k_2) = \max(f_{k_1}(\sigma) - f_{k_2}(\sigma), 0) \quad (5.10)$$

Conceptually this can be considered the frequency of state occurrences for a given play-styles (k_1) after subtracting the number of occurrences for that same state for another play-style (k_2). Once again this is similarly inspired by the complement operator in boolean algebra where $A \setminus B$ means everything in A that is not in B [Sikorski and others 1969]. Again for an intuitive depiction, we look to the example introduced by Figures 5.4 and 5.5. In this case, we apply Equation 5.10 to obtain the results depicted in Figure 5.8. Here it is important to note that since we are looking for everything in f_{k_1} that is not in f_{k_2} , we only need to consider the states that occur in both sets, as well as their frequencies. For example state $(2, 7, 1, 1)$ appears in both therefore we take its frequency in f_{k_1} and subtract its frequency in f_{k_2} . Since this results in a value ≤ 0 , the max of this respect to 0 will always be 0 and therefore this state won't appear in the resultant set of differing states.

$$\begin{aligned}
f_{k_1} &= \left\{ \begin{array}{l} [(2,7,1,1):1], [(2,8,1,1):2], [(3,3,1,1):2], [(3,4,1,1):2], [(3,5,1,1):2], [(3,6,1,1):3], [(3,7,1,1):1], \\ [(3,8,1,1):1], [(4,1,1,1):1], [(4,2,1,1):1], [(4,3,1,1):1], [(4,6,1,1):1], [(5,2,1,1):1], [(5,3,1,1):1] \end{array} \right\} \\
f_{k_2} &= \left\{ \begin{array}{l} [(2,4,1,1):2], [(2,5,1,1):2], [(2,6,1,1):2], [(2,7,1,1):2], [(2,8,1,1):2], [(3,5,1,1):2], [(4,5,1,1):1], \\ [(1,4,0,1):2], [(2,2,0,1):1], [(2,3,0,1):1], [(2,4,0,1):2], [(3,3,0,1):1], [(3,4,0,1):1] \end{array} \right\} \\
\max(f_{k_1} - f_{k_2}, 0) &= \left\{ \begin{array}{l} [(2,7,1,1):1-2], [(2,8,1,1):2-2], [(3,3,1,1):2], [(3,4,1,1):2], [(3,5,1,1):2-2], \\ [(3,6,1,1):3], [(3,7,1,1):1], [(3,8,1,1):1], [(4,1,1,1):1], [(4,2,1,1):1], \\ [(4,3,1,1):1], [(4,6,1,1):1], [(5,2,1,1):1], [(5,3,1,1):1] \end{array} \right\} \\
&= \left\{ \begin{array}{l} [(3,3,1,1):2], [(3,4,1,1):2], [(3,6,1,1):3], [(3,7,1,1):1], [(3,8,1,1):1], \\ [(4,1,1,1):1], [(4,2,1,1):1], [(4,3,1,1):1], [(4,6,1,1):1], [(5,2,1,1):1], [(5,3,1,1):1] \end{array} \right\}
\end{aligned}$$

Figure 5.8: Differing states observed between f_{k_1} and f_{k_2} in the simple example

Unique Characteristics

To determine the states which are unique to a particular play-style we solve for Equation 5.11. Here uniqueness emerges when a particular state occurs more often than the total frequency for that state in all the other clusters. Fundamentally, we are utilising the identical difference equation as delineated in Equation 5.10, albeit in relation to all other play-styles. Additionally, a state is said to be more or less unique when compared to other unique states based on the frequency of occurrences.

$$Unique(\sigma, k_1) = \max \left(f_{k_1}(\sigma) - \sum_{h=0, h \neq k_1}^k [f_h(\sigma)], 0 \right) \quad (5.11)$$

To illustrate the functioning of Equation 5.10, we once again intuitively employ our example. We build upon our previous examples by introducing f_{k_3} , which comprises four states, as depicted in Figure 5.9. Through this example, we demonstrate the emergence of unique states when we take the difference of f_{k_1} with respect to both f_{k_2} and f_{k_3} simultaneously. Specifically, if we consider the state $(2, 8, 1, 1)$, the calculation of $\max(2 - (2 + 3), 0)$ results in zero, and hence this state is not considered to be unique.

$$\begin{aligned}
f_{k_1} &= \left\{ \begin{array}{l} [(2,7,1,1):1], [(2,8,1,1):2], [(3,3,1,1):2], [(3,4,1,1):2], [(3,5,1,1):2], [(3,6,1,1):3], [(3,7,1,1):1], \\ [(3,8,1,1):1], [(4,1,1,1):1], [(4,2,1,1):1], [(4,3,1,1):1], [(4,6,1,1):1], [(5,2,1,1):1], [(5,3,1,1):1] \end{array} \right\} \\
f_{k_2} &= \left\{ \begin{array}{l} [(2,4,1,1):2], [(2,5,1,1):2], [(2,6,1,1):2], [(2,7,1,1):2], [(2,8,1,1):2], [(3,5,1,1):2], [(4,5,1,1):1], \\ [(1,4,0,1):2], [(2,2,0,1):1], [(2,3,0,1):1], [(2,4,0,1):2], [(3,3,0,1):1], [(3,4,0,1):1] \end{array} \right\} \\
f_{k_3} &= \left\{ \begin{array}{l} [(3,4,1,1):2], [(3,3,1,1):2], [(2,7,1,1):1], [(2,8,1,1):3] \end{array} \right\} \\
\max [f_{k_1} - (f_{k_2} + f_{k_3}), 0] &= \left\{ \begin{array}{l} [(2,7,1,1):1-2+1], [(2,8,1,1):2-2+1], [(3,3,1,1):2-2], [(3,4,1,1):2-2], \\ [(3,5,1,1):2-2], [(3,6,1,1):3], [(3,7,1,1):1], [(3,8,1,1):1], [(4,1,1,1):1], \\ [(4,2,1,1):1], [(4,3,1,1):1], [(4,6,1,1):1], [(5,2,1,1):1], [(5,3,1,1):1] \end{array} \right\} \\
&= \left\{ \begin{array}{l} [(3,6,1,1):3], [(3,7,1,1):1], [(3,8,1,1):1], [(4,1,1,1):1], [(4,2,1,1):1], \\ [(4,3,1,1):1], [(4,6,1,1):1], [(5,2,1,1):1], [(5,3,1,1):1] \end{array} \right\}
\end{aligned}$$

Figure 5.9: Unique states observed for f_{k_1} given f_{k_2} and f_{k_3} in the simple example

5.2.3.3 Identifying Play-style Decision Boundaries

Using state-based frequency analysis for unique state discovery allows us to understand the regions in which players of a particular play-style should occupy. However, this approach does not reveal what decisions a player will make during gameplay to reside in those locations. The limitation of utilising state-based frequencies for comparison is that this process ignores the temporal nature of trajectories. We incorporate this temporal knowledge into our play-style characteristic identification approach by locating the states which constitute the boundaries between play-styles as depicted in Figure 5.10. At these points, players decide to perform some action which is highly indicative of a particular play-style.

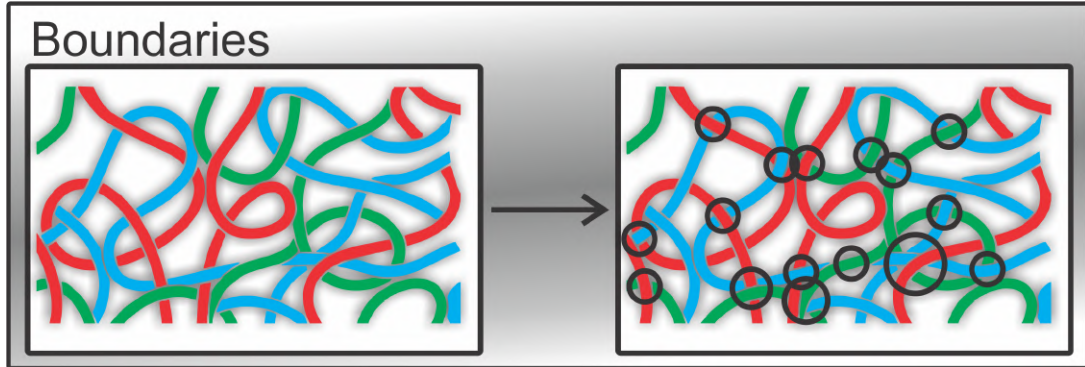


Figure 5.10: Example representation of the play-style decision boundaries identification process. Here decision boundaries are depicted as black circles

These points are determined using Algorithm 6 which works as follows: First, we create an empty dictionary *BoundDict* to store the decision boundaries and another empty dictionary *StateDict* to store the state-wise cluster prediction accuracies. We iterate over each trajectory X in our dataset ($\mathcal{D}$). For each state x_j in the trajectory, we create a set of partial trajectories P consisting of all states from the start of the trajectory up to state x_j . For each partial trajectory P_j , we calculate the prediction confidence using Equation 5.5 and identify the highest likelihood cluster prediction y' . We then update the dictionary *StateDict* with the average prediction confidence for each state and its corresponding cluster prediction y' . Once we have calculated the average prediction confidence for each state and prediction pair in our dataset, we iterate over every unique key in dictionary *StateDict*, which consists of tuples (x_i, y') representing each state and its highest likelihood cluster prediction. For each unique state x_i , we identify its neighbouring states $\mathcal{N}$ as any state which is a single action away. We calculate

the variance of the state-wise cluster prediction accuracies for all neighbouring states $x_j \in \mathcal{N}$ and store it in the variable *var*. If the variance *var* is above a given threshold v , we add the current state x_i to our decision boundary dictionary *BoundDict*. Therefore, a state is considered to be a boundary point if the variance between the confidence values of its neighbours exceeds a threshold parameter. One can intuitively regard this as any state where performing one action yields a high degree of confidence in the accuracy of the play-style prediction, whereas performing another action yields a low degree of confidence in the accuracy of the play-style prediction. Finally, we return the decision boundary dictionary *BoundDict* containing all identified decision boundaries.

Algorithm 6 Identifying Decision Boundaries

```

1: function IDENTIFYDECISIONBOUNDARIES( $D$ )
2:   Trajectory dataset  $\mathcal{D}$ 
3:    $BoundDict \leftarrow$  empty dictionary ▷ stores decision boundaries
4:    $StateDict \leftarrow$  empty dictionary ▷ stores state-wise cluster prediction accuracies
5:   for  $X \in D$  do
6:     for  $t \in \{0, 1, \dots, |X|\}$  do
7:        $P \leftarrow X[0 : t]$ 
8:        $\bar{k} \leftarrow$  prediction confidence of  $P$  ▷ using Equation 5.5
9:        $y' \leftarrow \operatorname{argmax}\{\bar{k}\}$  ▷  $y'$  : highest likelihood cluster prediction
10:      for  $x_j \in P$  do
11:         $StateDict[(x_j, y')] \leftarrow$  average of  $\bar{k}$  and  $StateDict[(x_j, y')]$ 
12:      for  $(x_i, y') \in$  unique keys of  $StateDict$  do
13:         $\mathcal{N} \leftarrow$  neighbouring states of  $x_i$  ▷ any reachable state one time-step away
14:         $var \leftarrow$  variance of  $StateDict[(x_j, y')]$  for all  $x_j \in \mathcal{N}$ 
15:        if  $var > v$  then
16:           $BoundDict[x_i] \leftarrow$  decision boundary
17:   return  $BoundDict$ 

```

Two examples of what constitutes a neighbouring state are depicted in Figure 5.11. Here we note that for the state (5, 6) regardless of the value of B_1 or B_2 the only neighbours are (5, 5) and (5, 7) since both (4, 6) and (6, 6) are non-traversable tiles. If we consider state (4, 2) we observe that all four cardinal directions are valid neighbours. If we were to consider a state (1, 4, 1, 1), a valid neighbour would be (2, 4, 0, 1). In this case, the value of B_1 has been updated as it has already been visited. However, (2, 4, 1, 1) would be an invalid neighbour in this scenario.

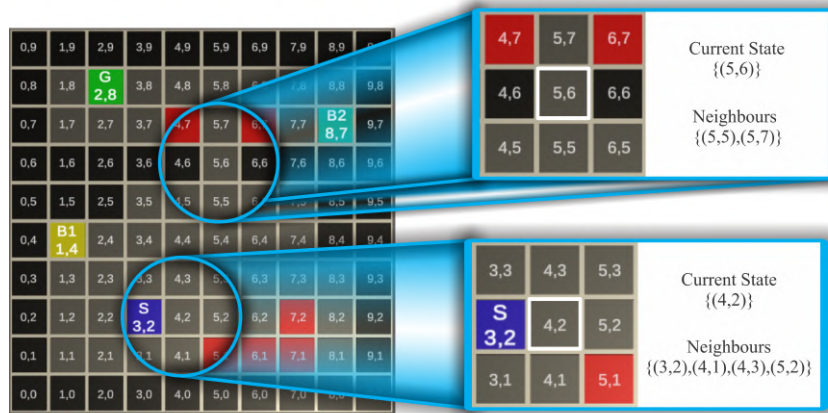


Figure 5.11: Example of neighbouring states for two states namely (5, 6) and (4, 2)

While not directly inspired, this approach demonstrates certain similarities with the research conducted by Kirman and Lawson [2009], who sought to model classification boundaries as abrupt changes in a social network game [Watts 2004]. By understanding these boundaries game designers can tailor specific segments of a player’s gaming experience to meet their play-styles specific needs and preferences.

5.2.3.4 Identifying Mean Play-style Behaviours

Mean play-style behaviour refers to a statistical representation of how a certain play-style is typically exhibited in a game. It can be determined by analysing data on the behaviour of multiple players who exhibit the same play-style, such as their decisions, actions, and other in-game behaviours, and determining the average or most common pattern of behaviour as represented in Figure 5.12.

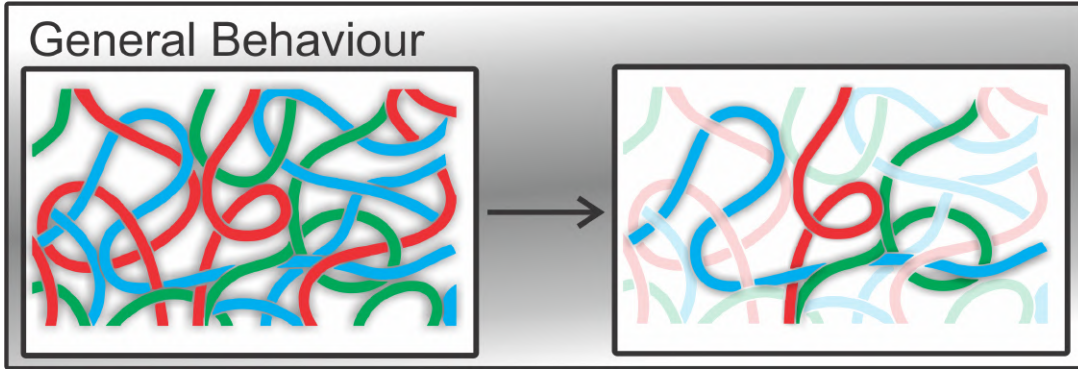


Figure 5.12: Example representation of the general play-style behaviours identification process

An added benefit of utilising our autoencoder clustering approach is the ability to utilise the identified centroids (C) for each cluster to determine examples of general play-style behaviours. We accomplish this in two ways, the first being to directly feed C_k in Equation 5.2 to reconstruct a mean trajectory C_k for each play-style k . This is possible since the centroids themselves exist within the generated latent space ($C \subseteq \mathcal{Z}$) of our network. Therefore, it can be reconstructed similarly to any other encoded vector. However, this does not guarantee that the trajectory is valid. Our second approach is to take the closest real encoded trajectory to the centroid and use that as the mean trajectory as determined by Equation 5.12. In this equation, $\|Z_i - C_k\|^2$ represents the squared Euclidean distance between the latent space representation Z_i and the specific representation C_k .

$$Mean(k) = \underset{Z_i \in \mathcal{P}}{\operatorname{argmin}} \sum_{i=1}^{n_k} \|Z_i - C_k\|^2 \quad (5.12)$$

This approach to generating mean play-style examples can be used to better describe the general expected behaviour for a particular style. Additionally, we can cluster new unseen trajectories using the mean trajectories for each style by applying the Longest Common Subsequence (LCS) algorithm [Needleman and Wunsch 1970]. We can modify the original algorithm for the longest common subsequence as presented in Algorithm 7, such that it takes two trajectories X_1 and X_2 as input, along with their lengths m_1 and m_2 respectively. The algorithm will then return the longest common subsequence of these two trajectories using a recursive approach by comparing each state of the trajectories on a one-to-one basis.

Algorithm 7 Longest Common Subsequence (Recursive)

```
1: function LCS( $X_1, X_2, m_1, m_2$ )
2:   if  $m_1 = 0$  or  $m_2 = 0$  then
3:     return empty sequence
4:   else if  $X_1[m_1 - 1] = X_2[m_2 - 1]$  then
5:     return LCS( $X_1, X_2, m_1 - 1, m_2 - 1$ ) +  $X_1[m_1 - 1]$ 
6:   else
7:      $L1 \leftarrow$  LCS( $X_1, X_2, m_1 - 1, m_2$ )
8:      $L2 \leftarrow$  LCS( $X_1, X_2, m_1, m_2 - 1$ )
9:     return  $L1$  if length of  $L1 \geq$  length of  $L2$ , else  $L2$ 
```

In this context, the trajectory being considered belongs to the play-style whose mean trajectory has the highest similarity score with the sample trajectory using the LCS algorithm. The determination of this is made by utilizing Algorithm 8, where X represents the sample trajectory that is being clustered, and C represents the set of all the play-style centroids.

Algorithm 8 LCS Clustering

```
1: function LCS-CLUSTERING( $X, C$ )
2:    $n = |X|$ 
3:    $L = 0$  ▷  $L$  represents the largest LCS value
4:    $k = 0$  ▷  $k$  represents the index of the largest LCS value
5:   for  $i \leftarrow 0$  to  $|C|$  do
6:      $c = C_i$ 
7:      $m = |c|$ 
8:      $T =$  LCS( $X, c, m, n$ ) ▷ as seen in Algorithm 7
9:     if  $L \leq T$  then
10:       $L = T$ 
11:       $k = i$ 
12:   return  $k$ 
```

5.2.4 Summary

Table 5.1 below serves to summarise all the previously discussed metrics as well as for which form of analysis they are associated.

Table 5.1: Summary of metrics used for each form of analysis

Section name	Section reference	Metrics used
Clustering Performance Analysis	Section 5.2.3.1	Equation 5.3 and Algorithm 5
Play-style Characteristic Identification	Section 5.2.3.2	Equations 5.9, 5.10 and 5.11
Decision Boundary Identification	Section 5.2.3.3	Algorithm 6
General Behaviour Identification	Section 5.2.3.4	Equation 5.12 and Algorithm 8

5.3 Training

Our model employs a two-phase training process, wherein the initial phase entails training an autoencoder architecture to obtain latent space representations, which are subsequently utilized for clustering purposes. We trained an individual model for each of our 7 environments ($E_1, \dots, E_5$, Mario, MiniDungeons).

5.3.1 Autoencoder

We employ a single-directional single-layered LSTM architecture for both the encoder and decoder network. The output sizes of the LSTM cells and latent vector representation were (20, 8) respectively. The output dimension of the encoder network as seen in Figure 5.1 is the size of our latent space. The input dimension is dependent on the dataset being used in particular ($E_1, \dots, E_5$) = 4, Mario = 6 and MiniDungeons = 15. To avoid dataset-specific tuning, we used the same parameters across all domains with ReLU as the activation function. The models were trained for 10000 episodes using the Adam optimiser with a learning rate 0.001 using mean squared error to determine the reconstruction loss as in Equation 5.13.

$$\text{Reconstruction Loss} = \frac{1}{|X|} \sum_{i=1}^{|X|} ||(X_i - X'_i)||_2^2 \quad (5.13)$$

5.3.2 Clustering

For the clustering step, we evaluated both k-means and Gaussian Mixture Models (GMMs). For both algorithms, the number of clusters were 4, 6 and 3 for the GridWorld, MiniDungeons and Mario domains, respectively. In the instances of the GridWorld and MiniDungeons domains, we opted to set the number of clusters to correspond to the actual number of policies employed in generating the domains. The purpose of this decision is to assess the effectiveness and behaviour of our model when the correct number of underlying play-styles is determined explicitly. In Section 5.4.5, we conduct experiments to confirm the accurate count of play-styles present. A similar experimental approach was used to determine the number of clusters in the Mario domain, where the actual number of clusters is unknown. For k-means and GMM, we fit our data using 100 restarts and a maximum iteration of 10000. In addition, k-means used a tolerance of 0.0001 and GMM used a “full” covariance type. k-means is a prototype-based method that assigns data points to the nearest cluster centroid, while GMM is a probabilistic model that represents clusters as Gaussian distributions. By employing both methods, we are exploring the diversity of clustering techniques and their ability to capture different structures in the data.

5.3.3 Baselines

To realise our model’s performance in terms of play-style identification we compare it against two baselines. The first is a random baseline which simply represents the chance of correctly predicting the corresponding play-style to a trajectory using a random policy. This is calculated as $\text{Accuracy} = \frac{1}{\text{Number of play-styles}}$. The second is a non-learning-based baseline by employing extensive data augmentation on the original dataset to facilitate subsequent k-means clustering. This procedure works by padding all trajectories with a standardised “zero” state achieving a uniform trajectory length. For example for the GridWorld environments, we pad with time-steps of (0, 0, 0, 0). Subsequently, we implement a custom RSME function to determine dissimilarity between pairs of trajectories. Both of these additions allow for the calculation and updating of the “k-means” as described in Section 2.6.1.

5.4 Results

Through the results of our experimental analysis, we demonstrate the ability of our model to accurately cluster game trajectories into their respective play-styles on both complete and partial trajectories. Additionally, we show how interpretive descriptions for each play-style can be recovered.

In Sections 5.4.1 and 5.4.2, we examine the performance of our model in comparison to the ground truth clusters, both offline and online. In Section 5.4.3, we investigate the results of characterizing play-style traits and their correlations with other play-styles. This leads to Section 5.4.4, where we illustrate our approach’s outcomes in identifying the decision boundaries between play-styles. Finally, in Sections 5.4.5 and 5.4.6, we present our discoveries regarding identifying the optimal number of clusters and general behavioural representations and how they can improve clustering performance.

5.4.1 Offline Trajectory Clustering

To demonstrate the effectiveness of our model in identifying play-styles, we plotted heat maps of the trajectories in each cluster as well as the original trajectories for each $r \in R$ for E_1 . In Figure 5.13 we calculated that there is a 65% similarity using Mean Absolute Error [Chai and Draxler 2014] between heat maps for both the clustered trajectories and the original trajectories separated by reward function.

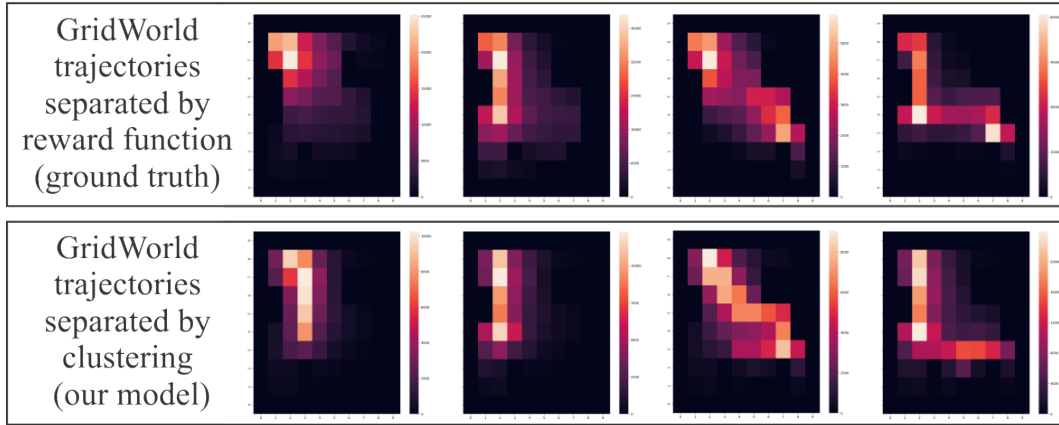


Figure 5.13: Heatmap of visited states comparing 8000 clustered trajectories separated by our model (bottom) and original trajectories separated by reward function (top) for E_1 .

This shows firstly, that the desired behaviours in Table 4.1 are represented in the data through the use of the rewards in Table 4.1. Secondly, we observe that the clustered trajectories depict the same behaviour. For example, R_4 sees the agent move to both bonus objectives (top-right in 5.13) and the same behaviour is observed in the corresponding clustered set (bottom-right).

For quantitative analysis, we directly compared the set of all predicted labels (y') with the set of all ground truth labels (y) for each Environment ($E_1, \dots, E_5$, MiniDungeons) using Equation 5.6. The clustering accuracies shown in Table 5.2 for both k-means and GMM clustering algorithms were obtained through our offline clustering procedure. Table 5.2 demonstrates our model’s ability to accurately cluster play-styles from completed trajectories across multiple varying environments. In this case, a complete trajectory represents $X \in \mathcal{D}$ whereas a partial trajectory represents $P = X[1 : t]$. In particular, we note, weaker performance within the MiniDungeons domain resulting from the higher complexity due to the dataset containing trajectories from multiple levels. In all domains, our performance surpassed that of a random policy that relied on clustering trajectories at random, based on a fixed number of clusters - specifically, four clusters for GridWorld and six for MiniDungeons. Additionally, we note

we compared against the non-learned approach, we can see that our model is able to better distinguish and identify different play-styles. In Table 5.2 we can see that the RSME baseline barely outperforms the random baseline. This discrepancy can be attributed to the substantial data augmentation necessary for handling trajectories through conventional clustering methods. This diverges from the data augmentation integrated into our model, where this process is acquired in a manner that preserves information integrity without introducing superfluous changes or information loss.

Table 5.2: Complete and partial trajectory clustering accuracy

Environment	Random Baseline	RSME Baseline	Offline GMM	Offline k-means	Online GMM	Online k-means
E_1	0.25	0.298	0.653	0.598	0.499	0.534
E_2	0.25	0.310	0.707	0.714	0.602	0.706
E_3	0.25	0.321	0.778	0.705	0.749	0.670
E_4	0.25	0.267	0.709	0.706	0.576	0.639
E_5	0.25	0.278	0.778	0.652	0.686	0.678
Average across GridWorlds	0.25	0.295	0.725	0.675	0.622	0.645
MiniDungeons	0.17	0.191	0.589	0.489	0.446	0.419

5.4.2 Online Trajectory Clustering

As previously mentioned, attempts have been made to cluster video game playthroughs [Drachen *et al.* 2009], however, very little work has been done in clustering trajectories online (*during gameplay*). To investigate the ability to identify play-styles during gameplay, we clustered partial trajectories and measured the change in clustering confidence as a function of time. This was achieved using Algorithm 5 with the results depicted in Table 5.2. The algorithm functions by initially encoding the partial trajectory and subsequently identifying the centroid that is closest to the encoding. The closest centroid corresponds to the cluster that the partial trajectory is most aligned with and consequently represents the corresponding play-style. We observe that an environment where trajectories are initially similar such as E_1 has a lower clustering performance. This indicates that the play-styles are initially well aligned while only diverging after some time.

In Figure 5.14 we observe that the cluster assignments are non-volatile after an initial fluctuation. However, two noticeable shifts occur, namely $3 \rightarrow 0$ which corresponds to the shift $R_1 \rightarrow R_2$ and $0 \rightarrow 2$ which corresponds to the shift $R_2 \rightarrow R_4$. This change in clustering assignment is valid as these trajectories are both from R_4 where the optimal behaviour was to move $B_1 \rightarrow B_2 \rightarrow G$. This corresponds with the expected behaviours described in Table 4.1. This also demonstrates that the autoencoder is separating data according to the reward functions. Therefore, this form of analysis can identify how a player’s play-style changes over time as a result of performing some objective in some fashion.

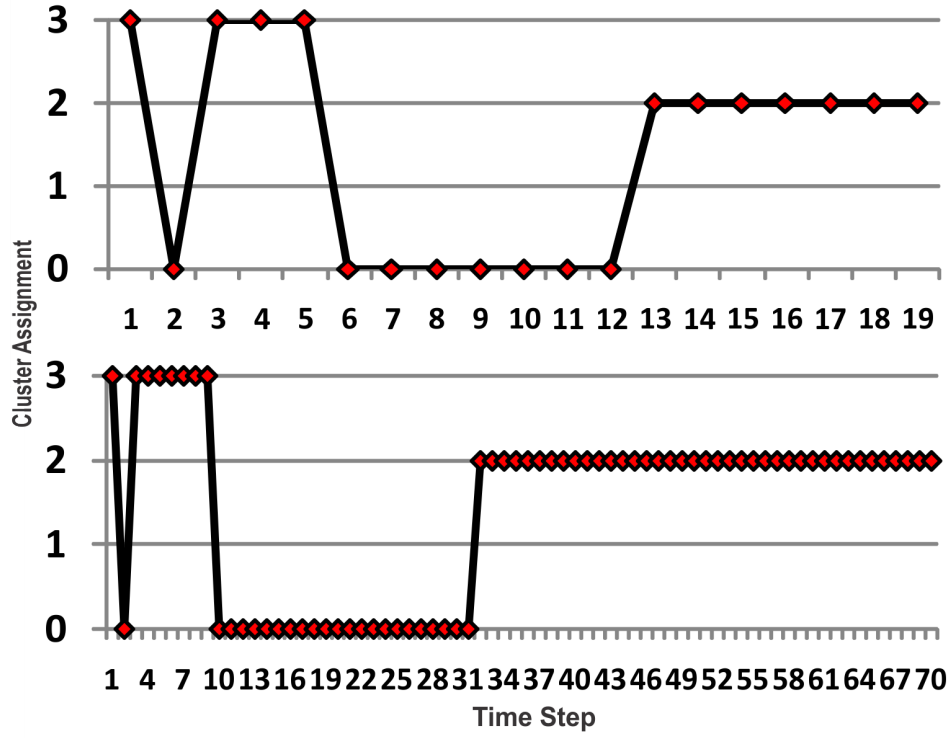


Figure 5.14: Change in play-style prediction over time for two particular trajectories with behaviour corresponding to R_4 with 2 being the correct cluster assignment for both

The image presented in Figure ?? illustrates the evolution of cluster assignments over time for two distinct trajectories in the Mario domain. The graph shows how these two trajectories eventually converge to the same play-style, despite starting with different behaviours. Trajectories in a video game are subject to change over time and can even converge due to shifts in the game’s dynamics, and our model is capable of capturing such changes effectively. The occurrence of fluctuations in the graph indicates regions of uncertainty in our model, which are suggestive of cross-over regions between play-styles. The primary advantage of identifying play-styles temporally is that it provides developers or players with a more detailed signal of when and where different styles become uniquely distinguishable.

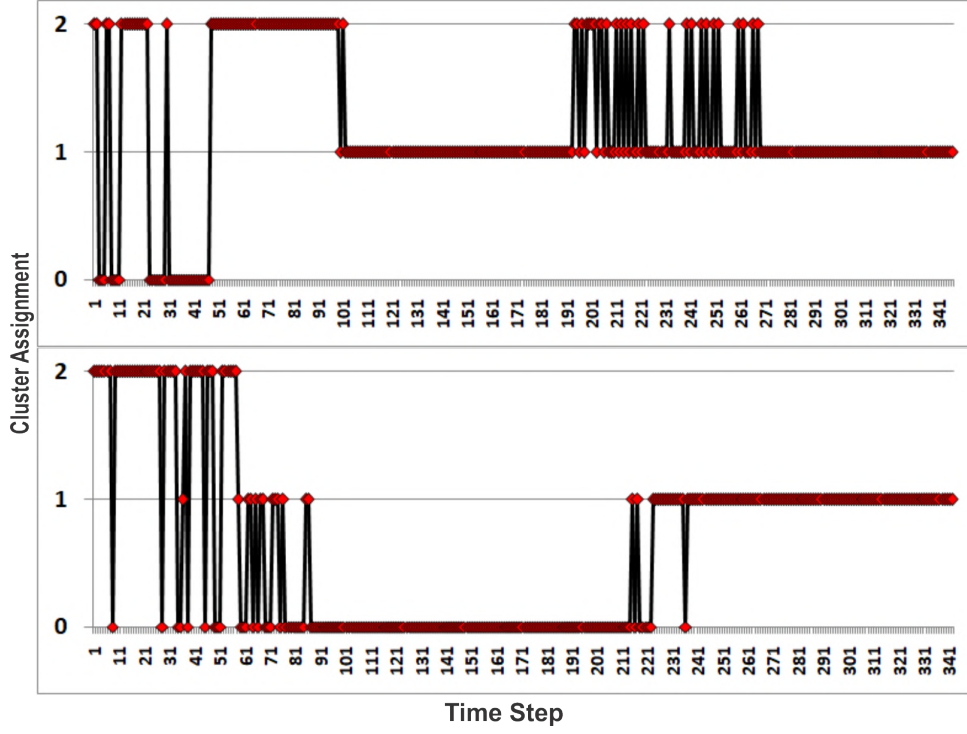


Figure 5.15: Change in play-style prediction over time for two particular trajectories from the Mario dataset

Both Figures 5.14 and 5.15 serve as behavioural examples of our online play-style identification process. To quantitatively demonstrate the efficacy of our partial clustering over time we analysed all trajectories separated by the identified clusters across all 5 GridWorld environments ($E_1, \dots, E_5$). This aggregation is possible since the play-styles we are trying to recover are the same across these different environments, as they would be across multiple levels in a video game. Figure 5.16 depicts the results of this, where we observe that for all environments the clustering assignment converges to a unique cluster. This demonstrates that across multiple trajectories our approach to partial trajectory clustering produces consistent desired results.

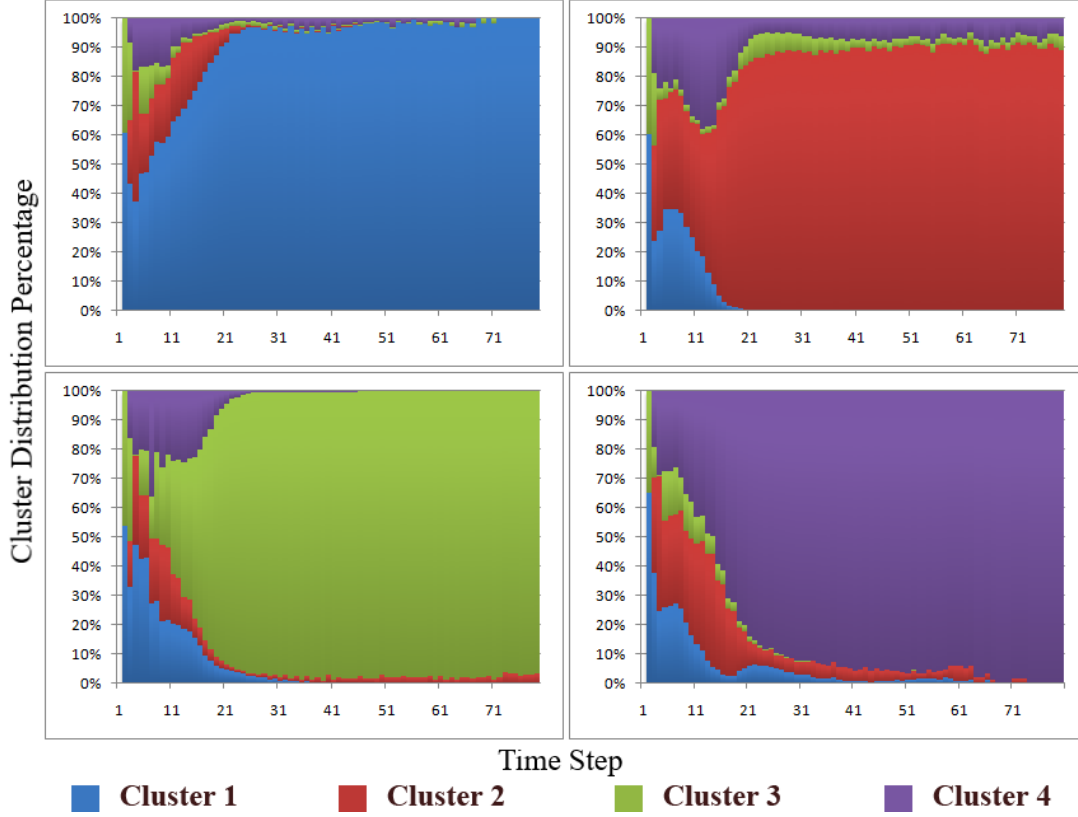


Figure 5.16: Clustering assignment over time separated by identified clusters over all 5 grid environments. Here each quadrant represents each of the generated partitions $\mathcal{P}$

The partial trajectory clustering accuracy (PTA), calculated using Algorithm 5, is shown in Table 5.2. The accuracy does decrease using partial trajectories, but not by a significant amount. This decrease, however, is to be expected with play-styles sharing similarities, most notably the start location. Taken together, these results demonstrate the robustness of the clustering model to unseen data within the domain as well as demonstrating our ability to quickly identify the correct play-style.

5.4.3 Identifying Play-style Characteristics

The results of applying Equation 5.8 outlined in Section 5.2.3.2 are visualised in Figure 5.17, where the state visitations are represented by a heatmap corresponding to the 4 different play-styles ($k = \{0, 1, 2, 3\}$). Our GridWorld trajectories are characterised by time steps that consist of four dimensions. Nonetheless, as described in Section 4.5.1, the bonus tiles are limited to two states, either 0 or 1. Hence, we can represent these four-dimensional states through the expansion of the following figures, taking into account the value of the bonus tiles (B_1 and B_2). Finally, we can reason about how each heatmap corresponds to the reward functions used to generate the data in the GridWorld environments as defined in Table 4.1. In particular cluster 1 (Figure 5.17a) strongly corresponds with R_4 and cluster 3 (Figure 5.17d) corresponds with R_3 . However, cluster 0 and cluster 2 share similarities with both R_1 and R_2 .

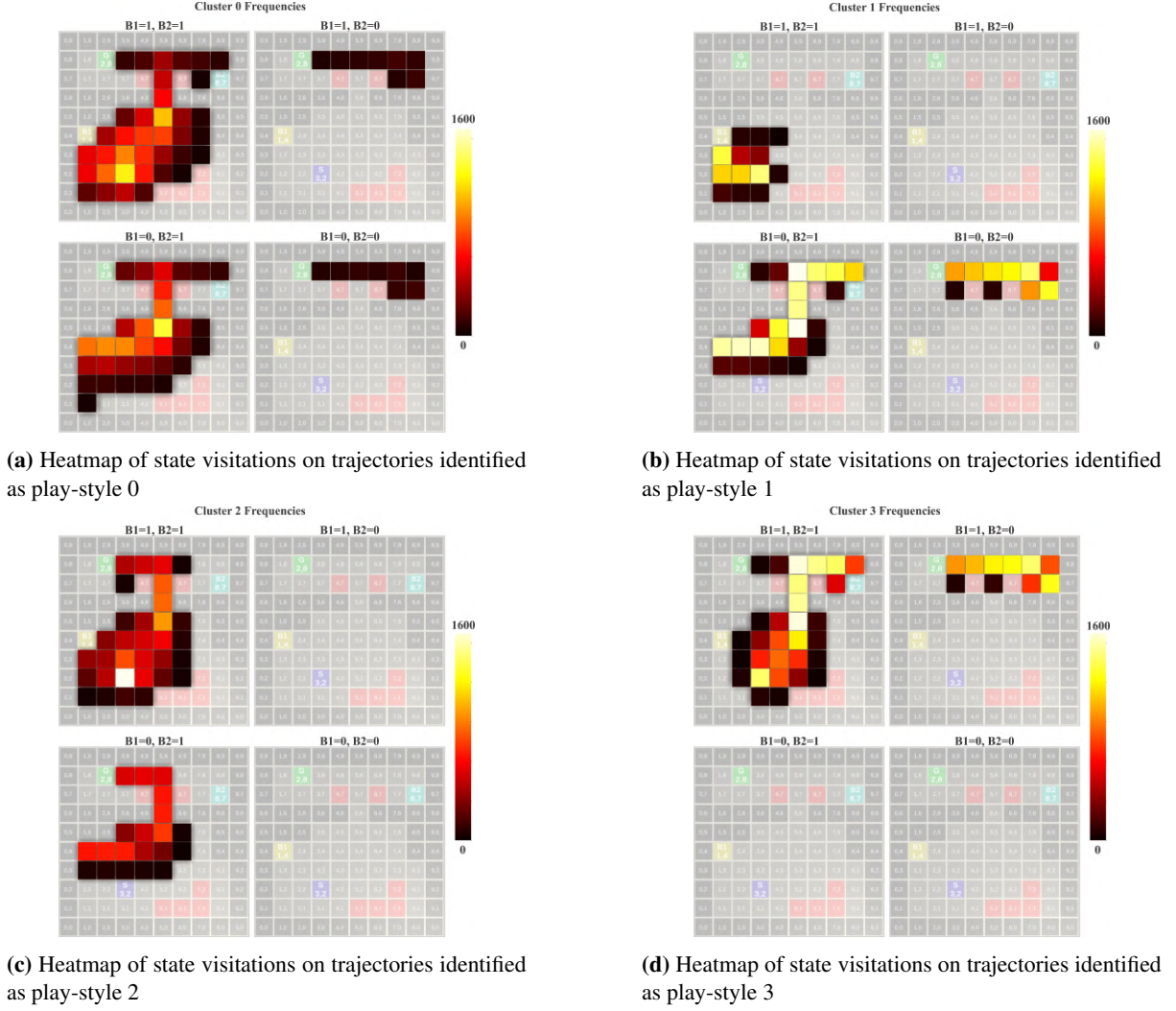


Figure 5.17: Collection of heatmaps across all trajectories clustered relative to four play-styles for E_4

Applying the same process to the Mario and MiniDungeons domain we obtain Figures 5.18 and 5.19 respectively. Here we observe that the frequency of the start state for the MiniDungeons domain dominates the rest, whereas this is not observed to such a degree leading to the belief that the Mario domain has a more diverse set of trajectories. However, through this visualisation, it is not easy to discern the behaviours of each play-style. Additionally, we note that as a result of the clustering process, it is not guaranteed that the number of trajectories across the clusters will be the same. This can cause issues in extreme cases where a cluster is populated by very few trajectories as the state frequencies can be dominated by another cluster. Therefore further analysis involving the relationships between play-styles is required. However through both Figures 5.18 and 5.19 we can observe the relationship between states and the differing play-styles. In particular we observe that different identified play-styles visit different states at vastly differing frequencies. This indicates that our state representation for the given domains is suitable for being able to separate differing behaviours. Lastly through this analysis we are able to visualise the most important states in the different domains across all play-styles, these being the states with the overall highest frequencies.

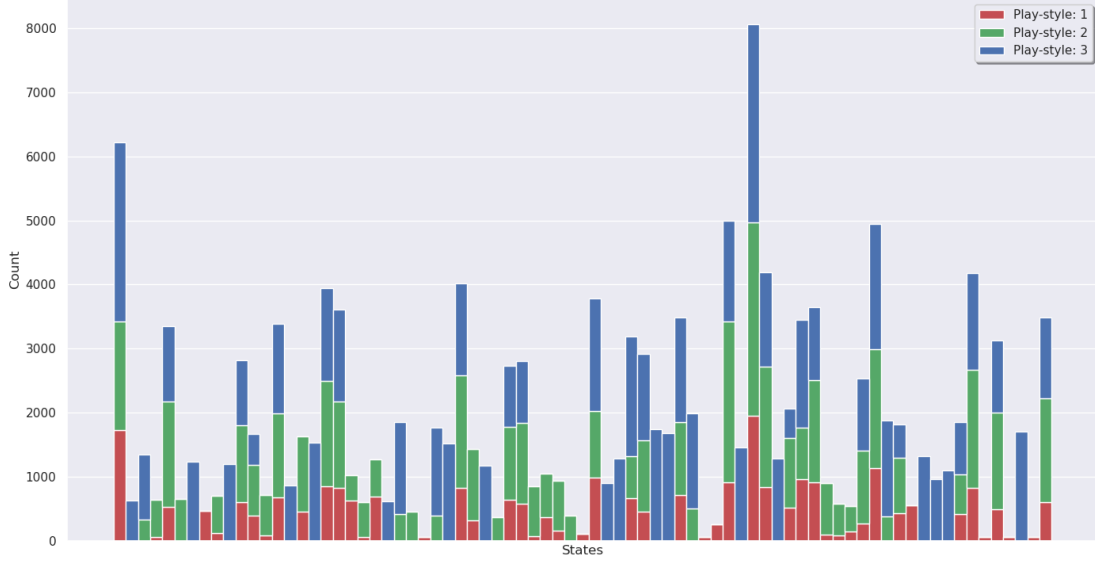


Figure 5.18: State visitations for the top 50 most common states across all play-styles contained within trajectories from the Mario domain

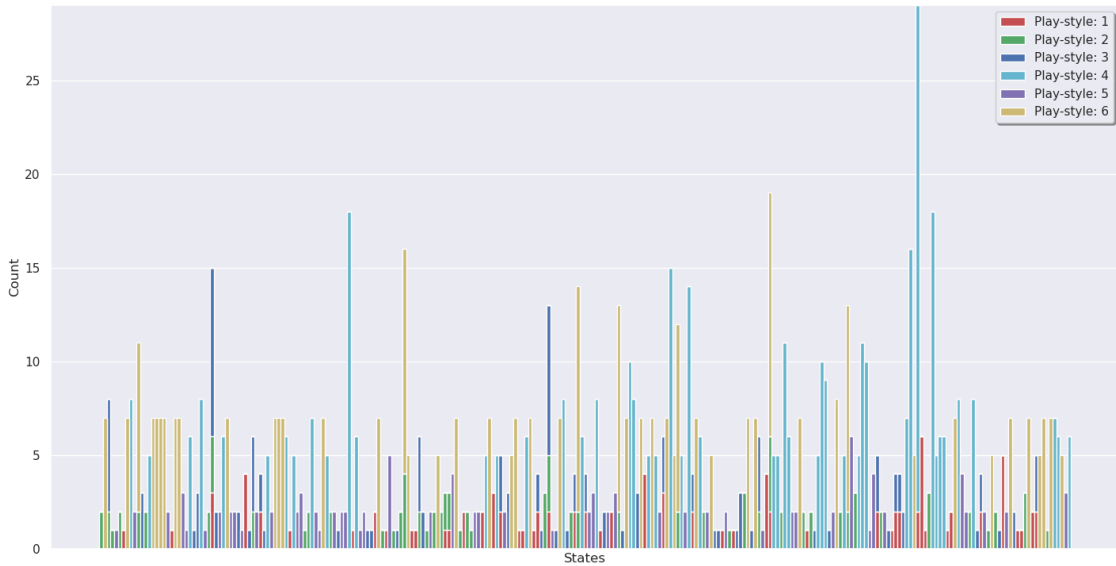


Figure 5.19: State visitations for the top 50 most common states across all play-styles contained within trajectories from the MiniDungeons domain

5.4.3.1 Shared States

For our Gridworld example, the overlapping states between clusters 1 and 3, are computed using Equation 5.9 as $Shared(\sigma, 1, 3)$ with results seen in Figure 5.20. Here it is observed that both play-styles initially congregate around B_1 . If we consider both $f_{k_1}(\sigma)$ and $f_{k_2}(\sigma)$ separately, which are also represented in Figure 5.20, we can observe that both heatmaps occupy the regions around B_1 . This observation validates our formulation of a “shared” state. We also observe that after obtaining B_1 which results

in the change in state to $B_1 = 0$, both play-styles occupy spaces within the narrow corridor in the middle of the environment.

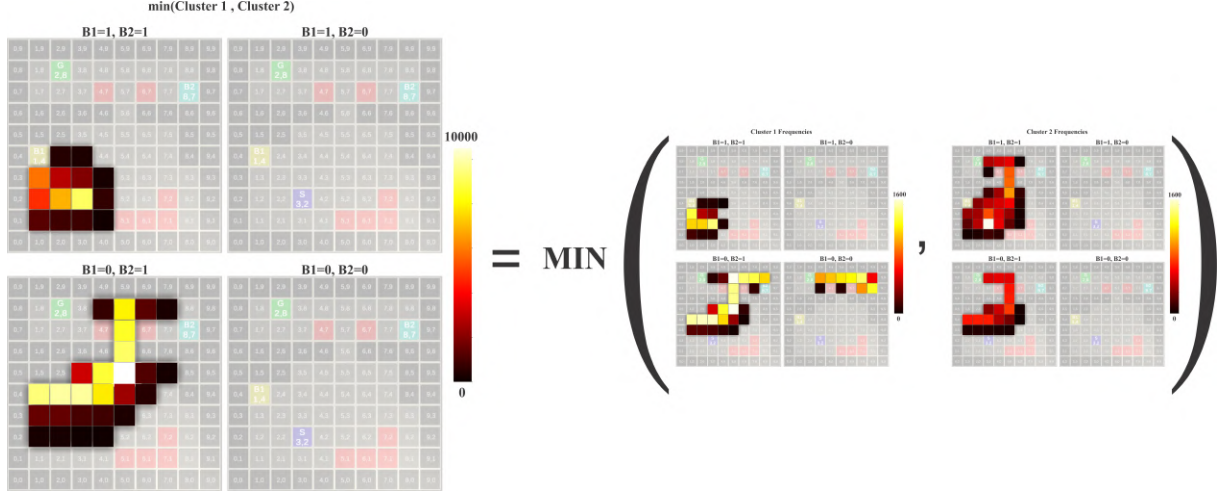
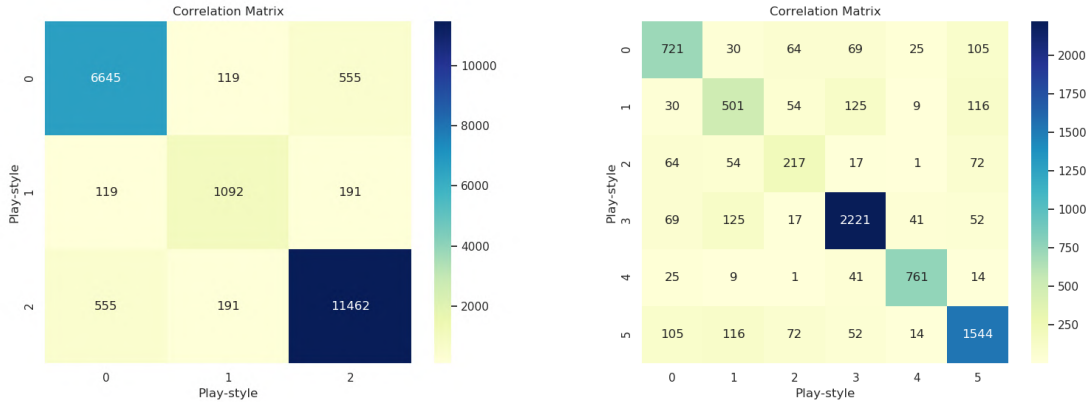


Figure 5.20: Shared states observed between Cluster 3 and 1 for E_4 (excluding non-visited states)

We conducted an analysis of the number of shared states for all possible play-style pairs of f_k in both Mario and MiniDungeons. Importantly this is just an indicator of the number of shared states and does not factor in the frequency of occurrences per state. Our findings are presented in Figure 5.21a and Figure 5.21b respectively. Notably, there is a strong correlation along the diagonal in both domains, which is an expected outcome when evaluating the similarity between identical play-styles. This significant correlation provides evidence that our model effectively separates trajectories into distinct clusters.

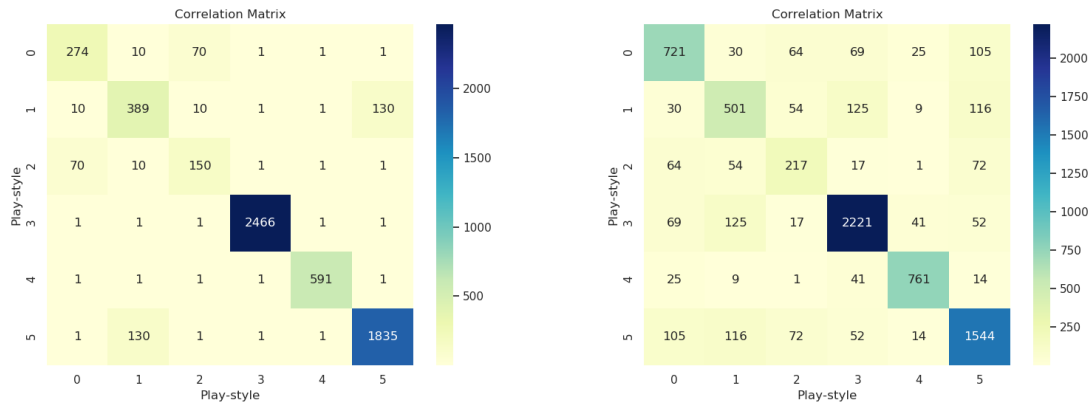


(a) Frequency of shared states observed relative to all play-styles for Mario (excluding non-visited states)

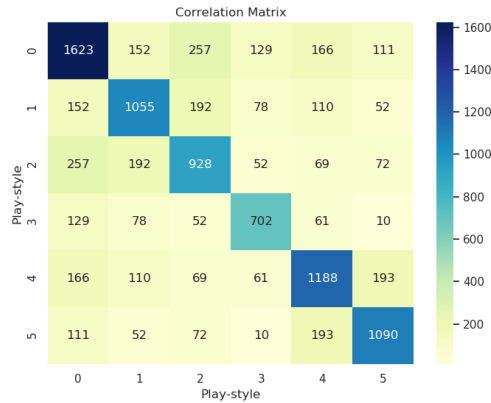
(b) Frequency of shared states observed relative to all play-styles for MiniDungeons (excluding non-visited states)

Additionally, we compared the shared states generated using our clustering methodology with that of the clusters separated using the ground-truth labels as observed in Figure 5.22. Once again using Mean Absolute Error we determine a similarity of 70% between matrices. This indicates that we can separate the trajectories into relatively similar partitions as those of the ground-truth labels. Lastly, Figure 5.22c illustrates a substantial increase in the overall quantity of shared states. As a result, we hypothesise that there is a direct correlation between a decrease in clustering accuracy and an increase in the total number

of shared states. This observation suggests that the optimal play-style clustering may be achieved by utilising a minimal set of shared states.



(a) ground-truth clustering (Total shared state frequency : 6167) (b) our unsupervised clustering method (Total shared state frequency : 7553)



(c) random clustering (Total shared state frequency : 9994)

Figure 5.22: Comparison between ground-truth, our method and random clustering in terms of shared states.

Equation 5.9 alone does not provide enough information to ascertain the point at which two play-styles deviate; for that purpose, we compute the disparity between the two play-styles.

5.4.3.2 Differing States

Equation 5.10 is used to identify the states of divergence as demonstrated here in Figure 5.23 which depicts the differing states between clusters 1 and 3. Here it is observed that the states surrounding B_2 are found in cluster 1 but not in cluster 3 which also corresponds with the expected behaviour. This behaviour is to go to the goal after obtaining B_1 rather than also going to obtain B_2 .

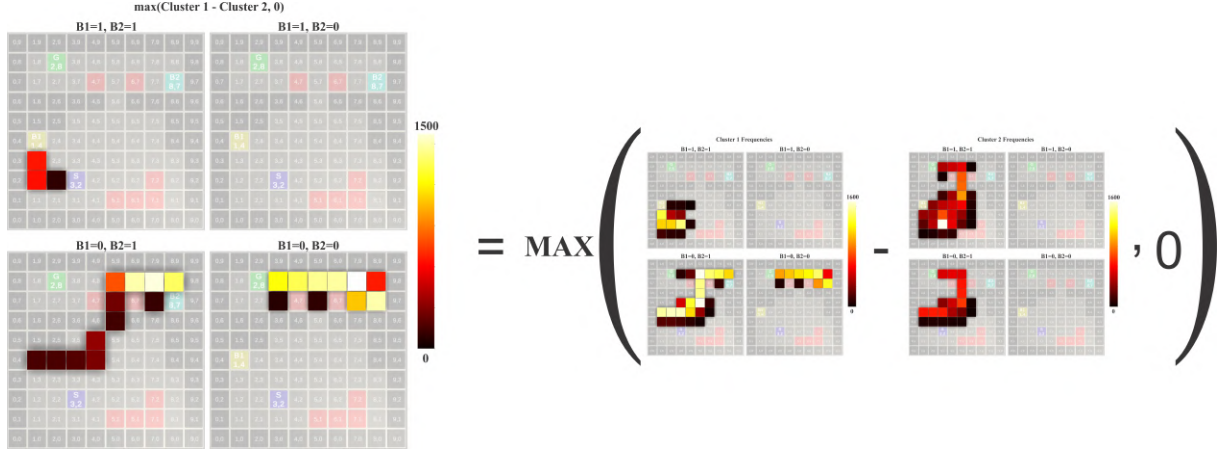


Figure 5.23: The States found in Cluster 1 without any of shared states with Cluster 3 ($\max(f_1 - f_3, 0)$)

It is important to note that Equation 5.10 is a relative difference calculation, as $\text{Diff}(\sigma, k_1, k_2) \neq \text{Diff}(\sigma, k_2, k_1)$. Therefore, the heatmaps generated using Equation 5.10 do not represent all the states not present in f_{k_1} or f_{k_2} . When comparing two play-styles, it can be informative to consider both the similarities and differences between the two states. However, such an analysis does not fully reveal the main features or characteristics of a specific play-style. To accomplish this, it is necessary to compare a given play-style with respect to all other play-styles to identify its unique characteristics.

In a similar fashion to the shared states the number of different states for all possible pairs of play-styles f_k in both Mario and MiniDungeons was analysed. It should be noted that this is solely an indicator of the number of different states and does not take into account the frequency of state occurrences. In the case of the Mario domain, as shown in Figure 5.24, we observe no differences along the diagonal, which is as expected when comparing a play-style with itself. This outcome could potentially serve as an indication that a sufficient number of clusters was selected. We expand on this notion in Section 5.4.5.

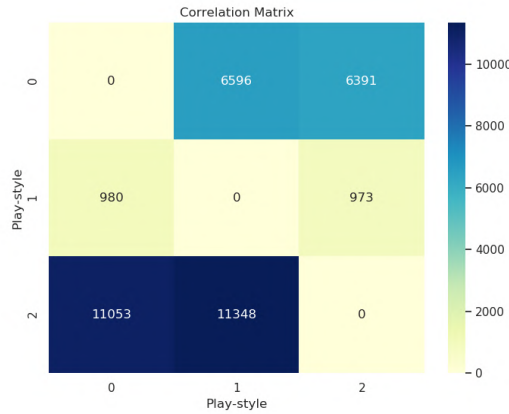
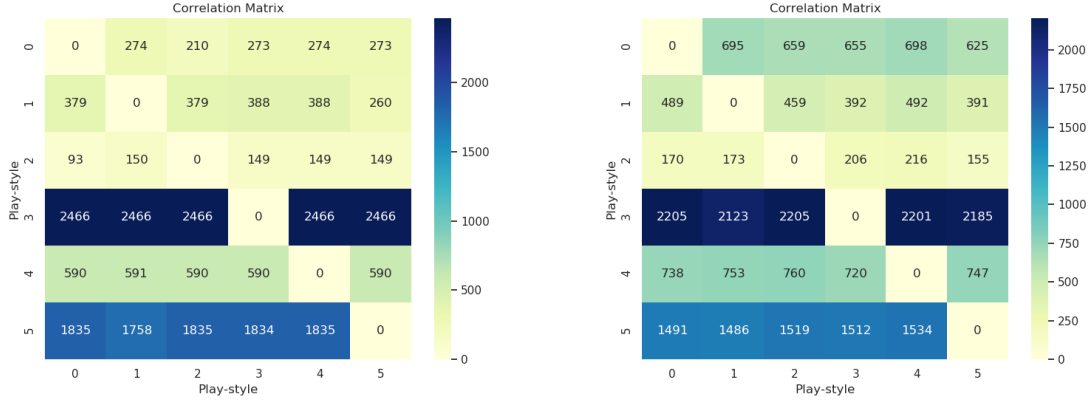


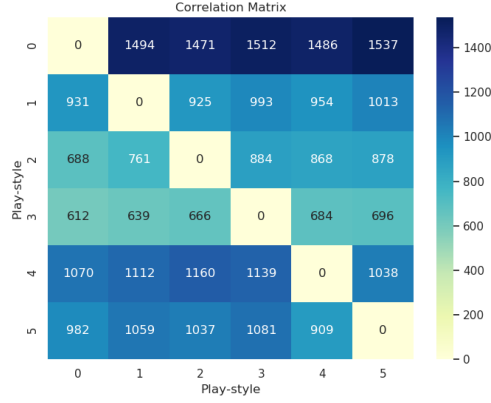
Figure 5.24: Frequency of differing states observed relative to all play-styles for Mario (excluding non-visited states)

Moreover, upon a comparison of the ground-truth correlation matrices and our approach depicted in Figure 5.25, we note that the outcomes are analogous to those observed in the shared states scenario. It is worth noting that play-styles three and five display considerable dissimilarity in comparison to all other play-styles. However, it should be acknowledged that this difference is relative ($\text{Diff}(\sigma, k_1, k_2) \neq$

$Diff(\sigma, k_2, k_1))$ and play-styles encompassing a vast number of diverse states will invariably result in elevated values as per Equation 5.10.



(a) ground-truth clustering (Total differing state frequency : 28166) (b) our unsupervised clustering method (Total differing state frequency : 28654)



(c) random clustering (Total differing state frequency : 30279)

Figure 5.25: Comparison between ground-truth, our method and random clustering in terms of differing states

5.4.3.3 Unique States

In particular, we observe in Figure 5.26 that the unique states for cluster 1 reside in the top right of the map for E_4 when $B_1 = 0, B_2 = 1$ as well as when $B_1 = 0, B_2 = 0$ which expectantly corresponds to behaviour 4 in Table 4.1. These unique states for cluster 1 ($k = 1$) were calculated as $Unique(\sigma, 1)$.

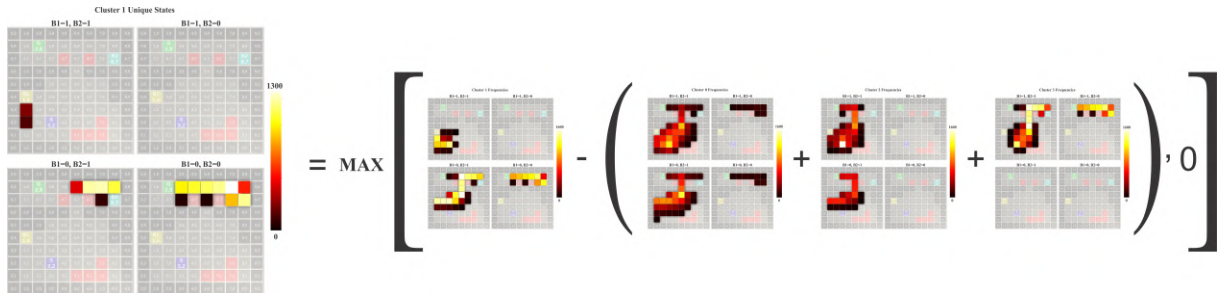


Figure 5.26: Unique states observed for Cluster 1 for E_4 (excluding non-visited states)

Identifying the unique characteristics of a play-style is important for an individual player in a game because it allows them to understand what sets their particular style apart from others. This knowledge can inform decision-making about game strategy, character selection, and in-game tactics, and help the player to develop their own distinct identity and approach to the game. Additionally, understanding the unique characteristics of a play-style can help a player to improve their gameplay by identifying their strengths and weaknesses, and allowing them to focus on the aspects of their play that are most effective. For example, a player that is known for their defensive play-style may be able to focus on improving their ability to read opponents and anticipate their moves, which can help them to win matches more consistently.

The application of the same analytical approach to the Mario dataset reveals the 50 most unique states across all play-styles, as depicted in Figure 5.27. It can be observed that “Play-style: 3” in the figure has the highest number of unique states, occurring at a significantly higher frequency than in other play-styles. In contrast, “Play-style: 1” has the lowest number of unique states, with the least degree of uniqueness for those states. However, it can be concluded that there is a set of states which can uniquely identify each play-style.

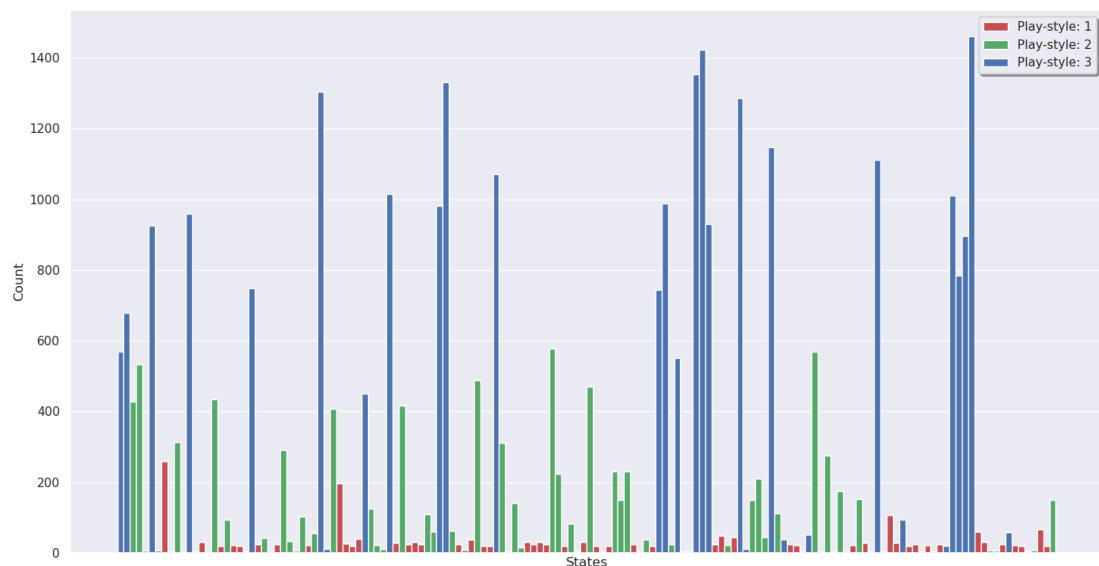


Figure 5.27: State visitations for the top 50 “most” unique states contained within trajectories from the Mario domain

Looking into the specific characteristics of the most unique states as seen in Table 5.3, allowed us to discover the meaning behind the identified clusters. For example, we note that cluster 0 corresponds to players who die the least while cluster 2 contains players who are more likely to collect coins while also killing enemies. By engaging in this behaviour we see that they are more likely to die. Lastly, cluster 1 players tend to jump the least while also ignoring picking up any coins.

The unique state counts can also be used to influence the number of clusters chosen when using k-means. By considering whether the frequency of the most unique state was too large or small we could raise and lower the number of clusters we identified. For example, if the frequency of unique states was below a certain threshold, the particular cluster would be considered too similar to another. By doing this we settled on 3 clusters that had both unique cluster characteristics and a suitable amount of data.

Table 5.3: The 3 most frequent unique states for each identified cluster for all the trajectories in the Mario dataset

	State Description					
	jumps	kills	runs	coins	deaths	action
Cluster 0	17	1	2	3	0	9
	17	1	2	3	0	2
	12	1	0	2	0	2
Cluster 1	11	7	0	0	1	9
	11	7	0	0	1	2
	8	7	0	0	1	2
Cluster 2	38	8	0	8	2	9
	38	8	0	8	2	9
	49	8	0	8	1	9

Finally, we determined the unique states associated with the MiniDungeons domain as seen in Figure 5.28. Here we observe similar characteristics as to both the GridWorld and Mario domains where each play-style is represented by a set of unique states. Additionally, we note that in particular “Play-style: 4” has the most frequently occurring and “most” unique states.

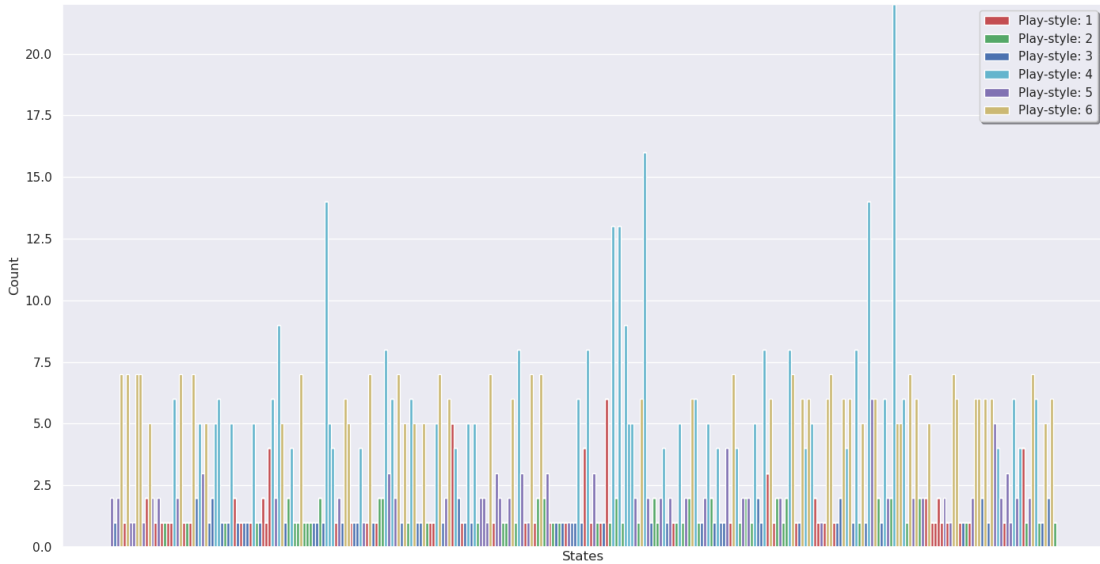


Figure 5.28: State visitations for the top 50 “most” unique states contained within trajectories from the MiniDungeons domain

By representing the unique states across play-styles we are able to formulate a general understanding of play-style behaviour as well as visually depict the diversity in behaviour.

5.4.4 Identifying Play-style Decision Boundaries

In Figure 5.29 we observe that when $B_1 = 0$, $B_2 = 1$ there is a transition boundary after moving up through the middle corridor where a player could then either go left or right. Here the play-style prediction accuracy greatly increases if one chooses to travel left while the accuracy for other actions is far less. This is per the behaviour of this particular cluster which is, as previously indicated, to obtain

B_1 and head to the goal. The converse can be observed in Figure 5.30 which represents the boundary points for cluster 2. For this case, the behaviour was to obtain B_2 followed by going to the goal. Once again after a player gets to the end of the corridor they need to decide whether to go left, right or back down, however, once the player decides to go right, the prediction accuracy greatly increases as opposed to following any of the other actions. As a result of this designers would only need to identify how a particular player acts around a small finite set of decision points to classify a player's behaviour.

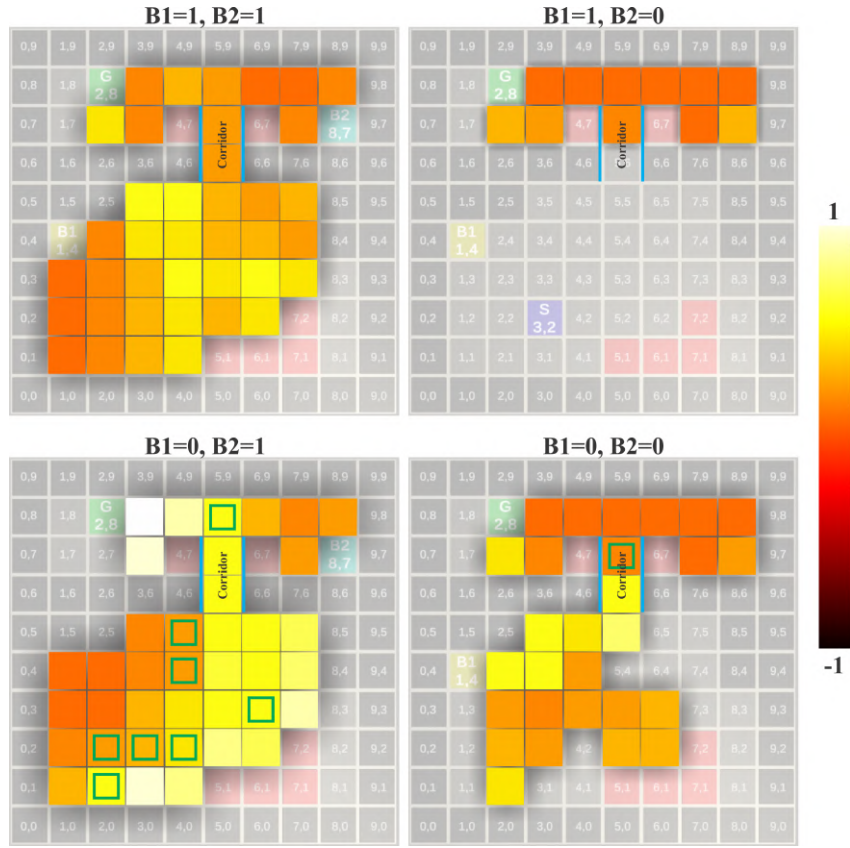


Figure 5.29: Play-style Decision Boundary Points indicated by green squares for cluster 1

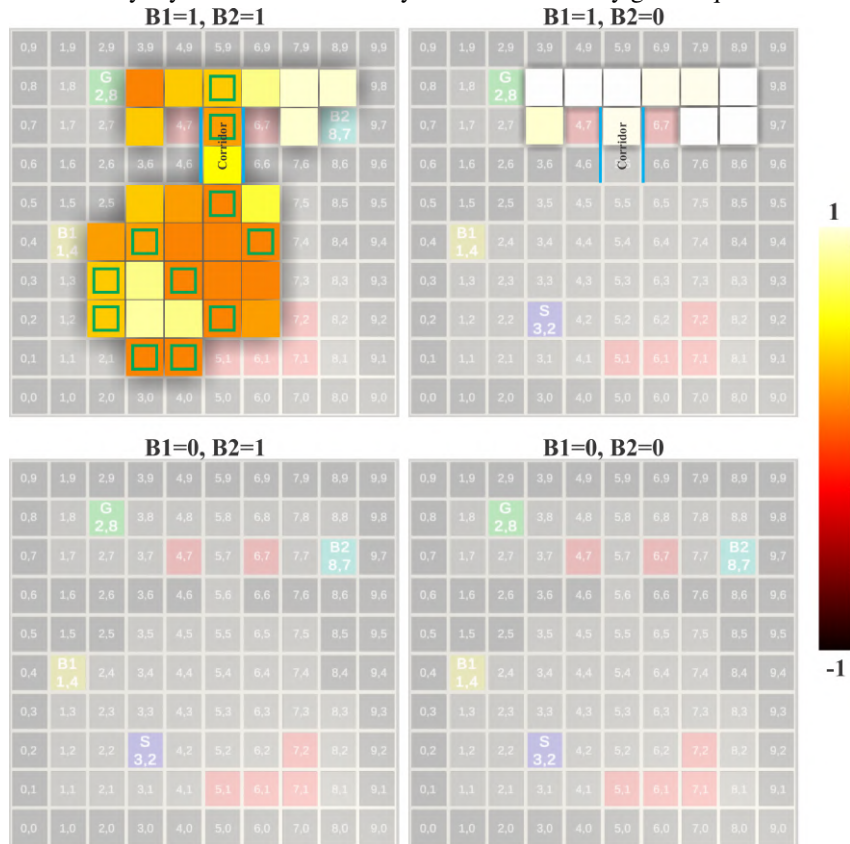


Figure 5.30: Play-style Decision Boundary Points indicated by green squares for cluster 2

Although these are just particular examples, the identified boundary regions for each GridWorld environment can be seen in Appendix B.1. Here we observe a general trend whereby we can identify the points where initially indistinguishable play-styles suddenly diverge as a result of one or two actions. However, our approach performs worse when play-styles gradually drift apart.

5.4.5 Determining the Appropriate Number of Play-styles

A limitation of our approach lies in the choice of k which corresponds to the number of play-styles that we aim to identify. By choosing an arbitrary k our resultant number of play-styles is not guaranteed to represent the true number of unique play-styles within a certain domain. This is a well-known issue related to both k-means and GMM approaches. A common solution, which has also been previously used [Arendse *et al.* 2022], has been to employ the ELBOW technique [Thorndike 1953]. However, we can utilise the identified shared, differing and unique characteristics of the clusters to determine if what has been identified has enough substance to be an individual cluster. Specifically, this is attained by computing the average number of unique states in comparison to the total number of clusters, and determining the point at which the slope of the curve ceases to decrease significantly. Figure 5.31 depicts the average uniqueness across all the identified play-style clusters for differing values of k calculated using Equation 5.14. Here it is observed that for GridWorld environments it is best to use four clusters to represent all the different play-styles, such that all clusters have a substantial number of unique states.

$$AverageUniqueness = \sum_{k=0}^n \left(\max \left[f_k(\sigma) - \left(\sum_{h=0}^n [f_h(\sigma)]; h \neq k \right), 0 \right] \right) \quad (5.14)$$

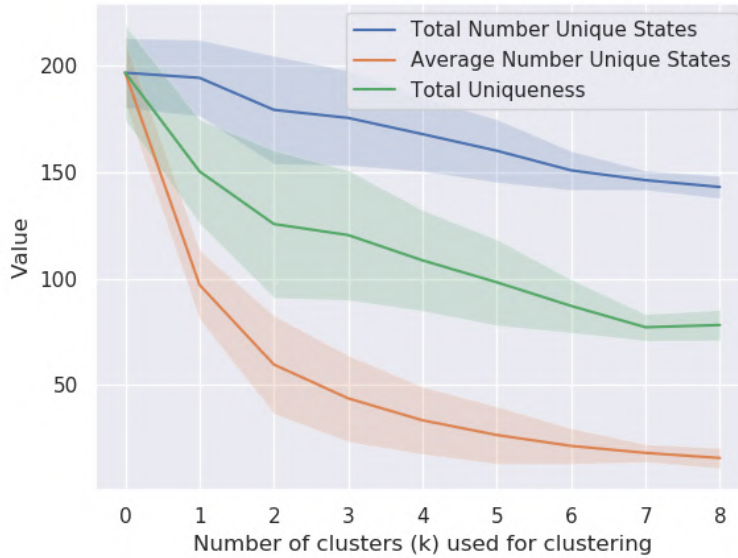


Figure 5.31: Analysis of uniqueness across all play-styles and all GridWorld environments for differing numbers of k

Although somewhat inconclusive we note that the appropriate number of clusters should fall between three and five according to this approach, whereas the ground truth number of clusters was four. This serves as an interesting exploration into the idea of using uniqueness for estimating the number of clusters in unsupervised methods, however, it requires further investigation.

5.4.6 Identifying Mean Play-style Trajectories

In Figure 5.32 we visualise the result of evaluating $Mean(k = 1)$ using Equation 5.12. In this case, we can see the generalised behaviour for this play-style is to head to B_1 then B_2 and finally to the goal. This directly corresponds to the expected behaviour described in Table 4.1 Visualising higher dimensions in this fashion is difficult, however, with this approach we can obtain state-transition dynamics representative of generalised behaviour for both Mario and MiniDungeons.

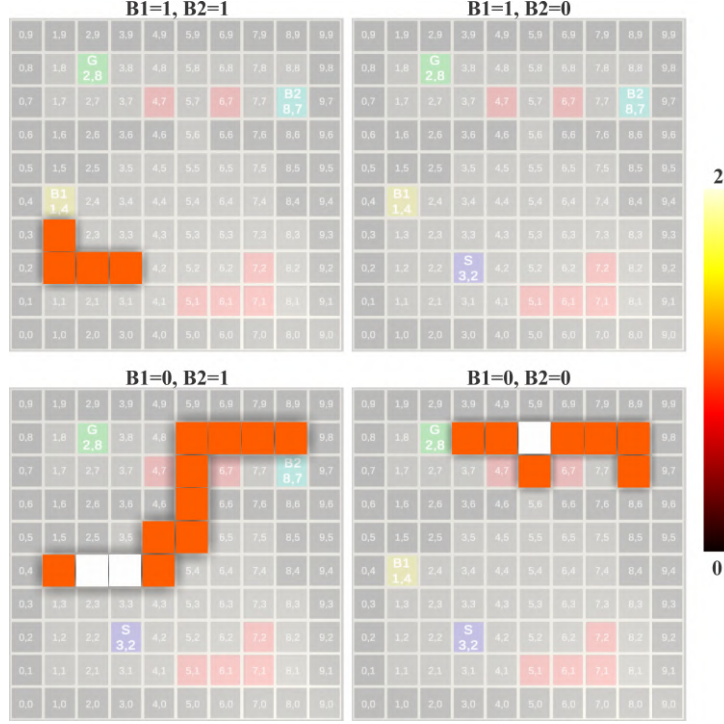


Figure 5.32: Mean Trajectory for Cluster 1 represented as a visitation map

By applying this method we obtained results seen in Table 5.4 comparable to that of our unsupervised approach, however in some cases, particularly for E_1 and E_5 we observed notably greater clustering accuracy. We note that the high play-style prediction accuracy is due to the mean trajectories for both E_1 and E_5 being highly correlated to the expected general behaviours for each play-style in those environments. The heatmaps for these mean trajectories can be seen in Appendix B.2. This leads us to believe that there is a benefit to utilising distance metrics which value structural similarity. Furthermore, this demonstrates that by identifying “mean” play-style representations we can improve the play-style prediction accuracy on unseen trajectories. It is important to acknowledge that our autoencoder model remains indispensable for generating the means, hence this strategy is not intended to supplant the model but rather to utilise it in conjunction to possibly attain superior outcomes.

Table 5.4: Comparison between LCS clustering with mean trajectories versus our autoencoder model

Domain	LCS Prediction Accuracy	Unsupervised Model Prediction Accuracy
E_1	0.598	0.653
E_2	0.644	0.707
E_3	0.931	0.778
E_4	0.643	0.709
E_5	0.893	0.778

5.5 Conclusion

In this chapter, we have presented a novel system for play-style identification that addresses the challenges of processing complex data and finding interpretive descriptions of behaviours within a dataset. By clustering multi-dimensional variable-length trajectories in video games, we have shown that our system can accurately identify different styles of behaviours exhibited by players. The core of our system is a specialised LSTM-autoencoder that can effectively handle sequence-based data by utilising the benefits of Recurrent Neural Network architectures.

Our system has several potential benefits, including its ability to identify play-styles in both offline and online settings without requiring additional engineering or training time. We have demonstrated this capability in Sections 5.4.1 and 5.4.2. Moreover, our system can recover the characteristics of each play-style through state-based cluster analysis in Section 5.4.3, which can provide useful insights for game designers to better understand player behaviour.

In addition, our system can identify the decision boundaries that separate different play-styles, as demonstrated in Section 5.4.4. This can help designers to create games that appeal to different play-styles and provide a more personalised experience for players. Finally, our process for generating mean representations of each play-style and determining the appropriate number of play-styles, as shown in Sections 5.4.6 and 5.4.5, can provide a more accurate and informative representation of player behaviour.

In summary, online identification of a player's play-style from partial trajectories during a live gameplay experience provides numerous benefits, including real-time adaptation, improved player engagement, dynamic matchmaking, and fraud detection and prevention. These benefits can lead to a more personalised, immersive, and enjoyable gaming experience for players, as well as a more successful game for developers.

Chapter 6

Game Modeller

Portions of this chapter have been published in [Ingram et al. \[2023a\]](#).

Branden Ingram, Benjamin Rosman, Richard Klein, and Clint van Alten. Creating Diverse Play-Style-Centric Agents through Behavioural Cloning. In Proceedings of the Eighteenth (AAAI) Conference on Artificial Intelligence and Interactive Digital Entertainment, (AIIDE) 2023.

6.1 Introduction

The development of diverse and realistic agents has become an important goal for game developers required to ensure the game world feels more immersive and believable, which can enhance player satisfaction and enjoyment. Traditional techniques in game design tend to involve hand-crafted solutions while in recent years approaches for learning game-playing agents often focus on optimising for a single objective, such as maximising the score or winning the game [\[Berner et al. 2019\]](#). While these agents may perform well in specific scenarios, they often lack the ability to adapt their play-style to different situations and opponents [\[McIlroy-Young et al. 2020\]](#). For example in chess, certain aggressive opening strategies can be countered by putting pressure on key points on the board, while more defensive opening strategies can be vulnerable to an opponent who seeks to control the board and develop their pieces effectively [\[De Marzo and Servedio 2022\]](#). Additionally, these agents may not be able to represent the full range of play-styles that human players exhibit, such as “speed-runner”, “completionist”, or unpredictable play-styles [\[Drachen et al. 2009\]](#). In particular, these types of play-styles could be advantageous to designers by serving as a form of playtesting, either independently or in conjunction with human players. Such an approach would enable a more informative development cycle by providing a deeper understanding of how the player base could react to various design decisions. In terms of players, utilising human-like agents maintains a level of realism and fairness not awarded by an optimal agent. There is some empirical evidence indicating that human players prefer playing with and against human-like agents [\[Arrabales et al. 2012\]](#). Therefore, the question becomes not how can we learn an optimal policy but rather how can we learn a set of policies which exhibit diverse behaviours or styles.

Despite the significant success achieved in developing singular optimal policies for multiple video game domains [\[Berner et al. 2019\]](#), there has been considerably less research focused on creating sets of agents exhibiting diverse behaviours. Alternative approaches that aim to replicate observed behaviours have demonstrated their effectiveness in generating human-like behaviours [\[Pearce and Zhu 2022\]](#). In the context of either of these approaches, learning varied policies relies on either designing or automatically identifying sets of characteristics from which a similar behavioural policy can be learned. To design a policy, we define relevant features that describe the desired play-style, and then we can learn a

policy that maps these features to actions [Arzate Cruz and Ramirez Uresti 2018]. Alternatively, we can automatically extract relevant play-style features from data using techniques like clustering or dimensionality reduction and then learn a policy from those, as demonstrated in Chapter 5. In both cases, the goal is to learn a set of play-style-centric agents whose behaviour matches that of either a designed or identified play-style.

Creating such a set of play-style-centric agents that play according to a certain style can have several potential benefits. Firstly, by generating a group of agents who play according to different styles we can provide players with a more diverse and varied gaming experience, as they will be playing against opponents with different strengths and weaknesses. Moreover, such a technique that generates a set of agents that adhere to a particular gameplay style can offer a cost-efficient alternative for supplying a greater quantity of adversaries for players to confront, in contrast to the conventional design methodology. Lastly, play-style-centric agents can be useful for players who want to learn more about a particular strategy or play-style, as they can observe the agents in action and analyse their decision-making processes. For example, a chess player might find enjoyment in being able to play against an opponent of a particular style or in training it may be useful to train against players of different styles.

In this chapter, we propose a novel pipeline which aims to generate a diverse set of play-style-centric agents dubbed PCPG (Play-style-Centric Policy Generation). This pipeline consists of a combination of unsupervised play-style identification and policy learning techniques. Through the unsupervised identification process, the demonstrations are divided into subsets based on identifiable behaviours. The policy learning method is then applied to train agents that can mimic a particular play-style. The ultimate goal is to create a diverse set of agents that can accurately represent various play-styles in the game. We evaluate the efficacy of our proposed approach on multiple video game domains and demonstrate that the agents generated by our pipeline can effectively capture the richness and diversity of gameplay experiences. Our results show that our agents learn policy with a high degree of accuracy while showcasing identifiable and diverse play-styles.

6.2 Methodology

We aim to solve the following problem: given a set of trajectories ($\mathcal{D}$) can we separate them into k distinct partitions ($\mathcal{P}_k$) from which we can generate a policy (π_k) which exhibits the behaviour identified for each partition identified by k ? Furthermore, can we further partition $\mathcal{P}_k$ such that we can learn optimal policies (π_k^*) for each play-style k ?

6.2.1 Play-style-Centric Policy Generation (PCPG)

Our PCPG system as depicted in Figure 6.1 is based on three key steps. In step A, we use a clustering technique to separate our observations, or in this case our multi-dimensional trajectory data, according to different identifiable play-styles as demonstrated by the PI in Chapter 5. In step B, we order the trajectories based on a ranking function. Step C is then to train individual Behavioural Cloning (BC) models on a selection of the top-performing observations for each of the generated partitions. Since each of these selections contains data associated with a specific behaviour, it is expected that the learned policies would display similarities to that behaviour. Therefore, these models trained using behavioural cloning are referred to as play-style-centric models.

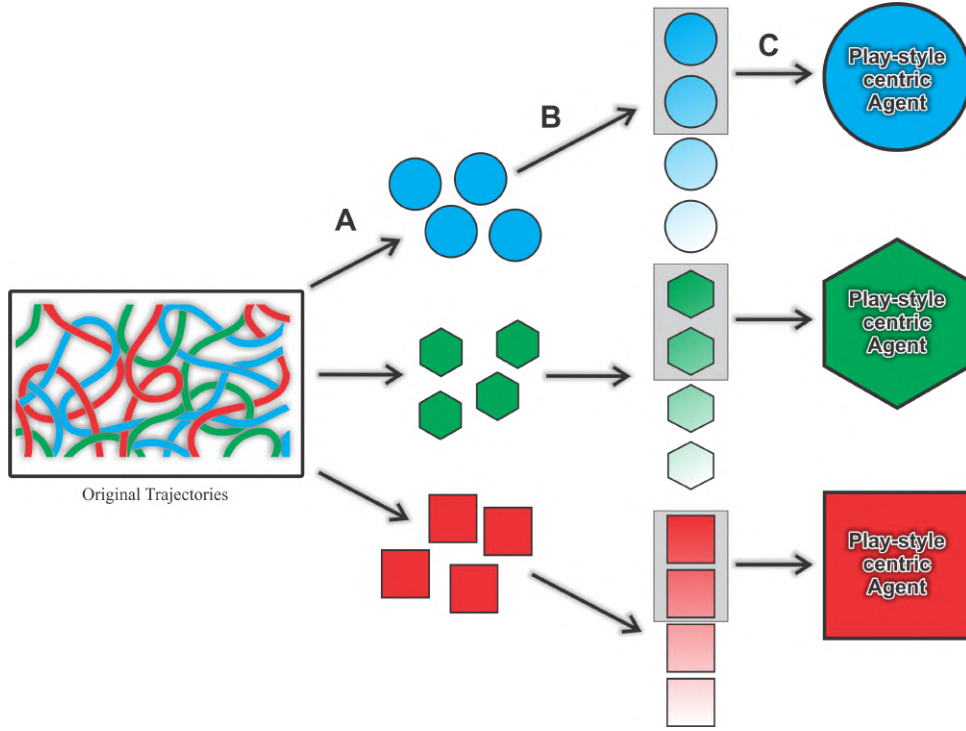


Figure 6.1: Overall PCPG system; initially the set of trajectories is clustered in step A into subsets H_k , each subset is then rank-ordered in step B indicated by the colour gradient, and finally the top performing trajectories are thresholded ($\mathcal{P}_{k,p}$) and used to train the play-style-centric agent in step C.

6.2.1.1 Trajectory Clustering (Step A)

In the context of this chapter, we use the PI to perform the task of separating trajectories into partitions with respect to the style. To recap, this works by first projecting a trajectory ($X_i \in \mathcal{D}$) into a lower-dimensional latent representation ($Z_i \in \mathcal{Z}$) using an LSTM autoencoder network. Subsequently, we cluster this latent space ($\mathcal{Z}$). This process aims to discover k clusters, resulting in the creation of k partitions ($\mathcal{P}$) of the original trajectories.

6.2.1.2 Policy Learning (Steps B and C)

Next, we look to train representative policies having clustered all trajectories ($X \in \mathcal{D}$) into separate subsets ($\mathcal{P}_k \subset \mathcal{D}$) where k is the cluster identifier. In step B each $\mathcal{P}_k$ is reordered based on a ranking metric. Subsequently, a thresholding parameter (p) is used to choose a subset of trajectories ($\mathcal{P}_{k,p} \subseteq \mathcal{P}_k$) that perform in the top “ p ” percentile, which will be utilised as inputs for the corresponding play-style-centric model. This is implemented to guarantee that the trajectories received by our play-style-centric models correspond to high-performance instances of the particular play-style. Lastly, in step C, the play-style-centric model is trained through a supervised learning approach that minimises the loss between the predicted action by the model and the expected action.

6.2.2 Training

For the trajectory clustering, we used an unsupervised LSTM clustering model as described in Chapter 5. Our play-style-centric models were made up of 3 fully connected layers of 5 nodes each. The dimensions of the input and output for this model correspond to the length of a single timestep, which is equal to

4 for the GridWorld domain and 15 for MiniDungeons. Once more we define k as 4, 6 and 3 for the GridWorld, MiniDungeons and Mario domains, respectively.

All experiments were conducted using 4 seeds across the 5 GridWorld environments as well as the Mario and MiniDungeons domains with the average accuracy for each being recorded. We trained each play-style-centric model for 2000 episodes. The initial dataset ($\mathcal{D}$) contains 4000 trajectories for each GridWorld environment 780 for MiniDungeons and 74 for Mario. For our ranking metric, we utilised Equation 6.1;

$$\text{Performance}(X) = \frac{1}{|X|} \cdot \begin{cases} 1 & \text{if } x_T \text{ is a goal state} \\ \beta & \text{otherwise} \end{cases} \quad (6.1)$$

where x_T denotes the final state in the trajectory X , and β is a constant penalty factor that reduces the score when x_T is not a goal state. This performance metric incentivises shorter trajectories by utilising the length of a trajectory calculated as $|X|$. Here higher values of $|X|$ correspond to lower performance, while smaller values of $|X|$ indicate better performance. Additionally, we adjust this metric based on whether a trajectory successfully reaches the goal state or not. For example, if $\beta = 0.5$, then the score is halved when the trajectory does not end in a goal state. In particular, we used $\beta = 0.1$ heavily penalising all trajectories which did not reach the goal state. We further demonstrate the functioning of this performance metric in Table 6.1, where we represent a few example scenarios for different values of the different parameters.

β	$ X $	is x_t a goal state	Performance
0.1	10	True	0.1
0.1	10	False	0.01
0.1	100	True	0.01
0.1	100	False	0.001
0.5	10	True	0.1
0.5	10	False	0.05
0.5	100	True	0.01
0.5	100	False	0.005

Table 6.1: Example scenarios for input values used to calculate the performance of a trajectory X as used in Equation 6.1

While in our domains we employ trajectory length as a performance measure, it is equally feasible to substitute it with a predefined score metric inherent in the environment or a different combination of features that can serve as indicators of overall skill. To showcase the advantage of “step C” in our approach, we trained an individual model with the same architecture as each play-style-centric model without clustering, as a baseline approach. Lastly, we tested over a range of thresholding percentages, in particular, where $p = [1, 0.75, 0.5, 0.25, 0.1]$. Here a value of 1 indicates we train our play-style-centric models on all trajectories while 0.1 indicates we train each model on the top 10% of trajectories.

6.2.3 Evaluation

In this study, model accuracy was determined by evaluating whether the model was able to correctly predict the action necessary to transition the agent from state s_i to s_{i+1} . Now, autoregressively predicting the next state at each step, where the model receives its previous outputs as inputs, would be problematic due to compounding errors. For instance, if the model makes a slight mistake at the start of the

trajectory, the final state would likely be very inaccurate. To remedy this and ensure that each prediction can be made independently of historical mistakes, we input, at each step, the ground truth state to the model instead of its previous predictions. This is a widely-used technique in language modelling and model-based reinforcement learning [Moerland *et al.* 2023]. This process is completed for all states in a trajectory and across all trajectories in the testing set.

6.3 Results

Through the results of our experimental analysis, we demonstrate the ability of our model to accurately represent desired behaviours which correspond to identified play-styles.

6.3.1 PCPG model performance

Figure 6.2 demonstrates that our PCPG model not only yields high performance but also surpasses the equivalent “No Clustering” baseline. Comparable outcomes can be observed for MiniDungeons depicted in Figure 6.3 and Mario in Figure 6.4. This benefit can be observed across all tested values for the performance threshold (p). The observed stability of our model in the face of alterations to the performance threshold underscores its robustness in accurately depicting a wide range of behaviours associated with different skill levels. This resilience implies that the model’s predictive capabilities remain consistent even as we increase the variety of skill levels present in our dataset. As a result, it becomes evident that our model possesses the capacity to effectively generalize and encapsulate the nuances of behaviours exhibited by individuals of varying skill proficiencies.

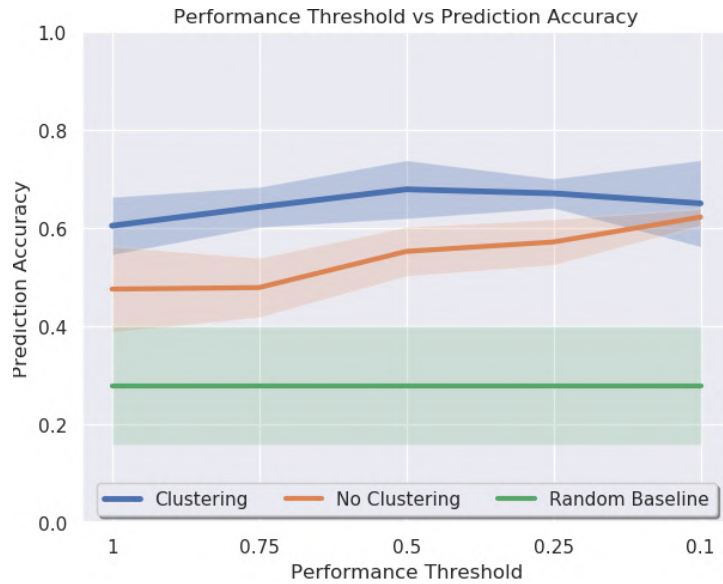


Figure 6.2: Comparison of the effect of differing the performance threshold percentage p on the model’s accuracy across all GridWorld environments ($E_1, \dots, E_5$)

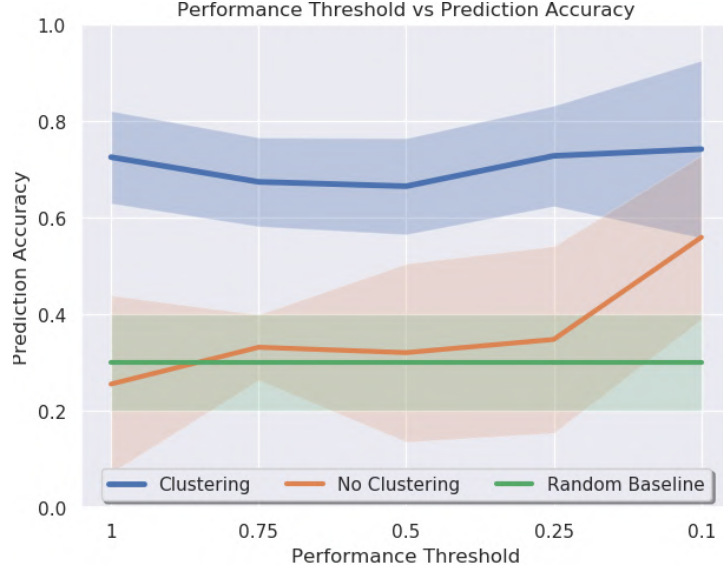


Figure 6.3: Comparison of the effect of differing the performance threshold percentage p on the model’s accuracy in MiniDungeons

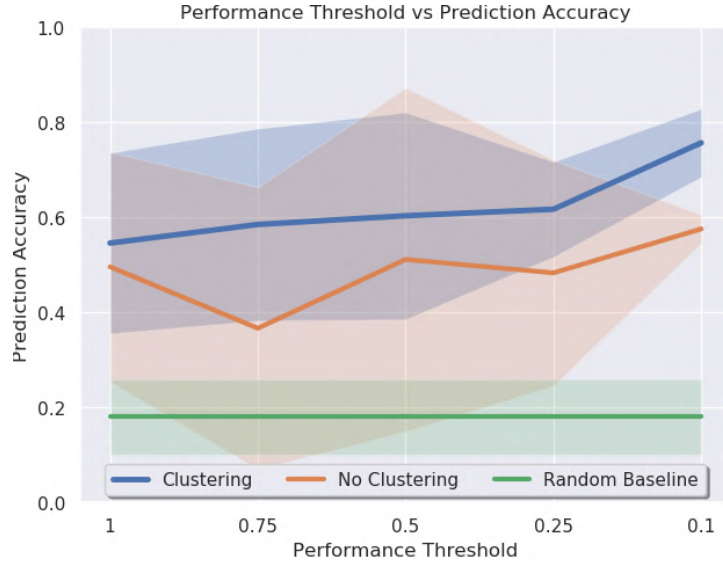


Figure 6.4: Comparison of the effect of differing the performance threshold percentage p on the model’s accuracy in Mario

It is important to mention that when using smaller subsets ($\mathcal{P}_{k,p}$) for training (lower p values), the performance of the “No Clustering” approach does improve. However, this improvement is mainly because there is a decrease in the variety of trajectories that are present in the training and testing sets. In contrast, our PCPG approach is not affected by this issue. This can be seen in Figure 6.5 where we depict the average trajectory diversity across p . Here we measure diversity based on the variety of lengths of the trajectories found in each dataset. We can see that, unlike the “No Clustering” approach, the trajectory diversity for our PCPG model does not converge to a single length. In Figure 6.6 we depict the diversity of trajectory lengths in the Mario domain. Here it is observed that once again as the performance threshold tends to 0.1 (indicating higher proficiencies), that each play-style tends to a different trajectory length.

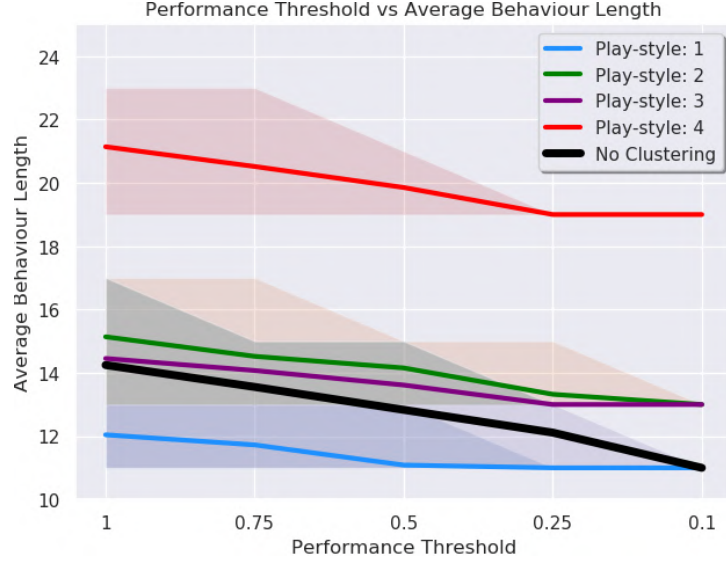


Figure 6.5: Comparison of the effect of differing the performance threshold percentage p on the trajectory diversity across all GridWorld environments ($E_1, \dots, E_5$)

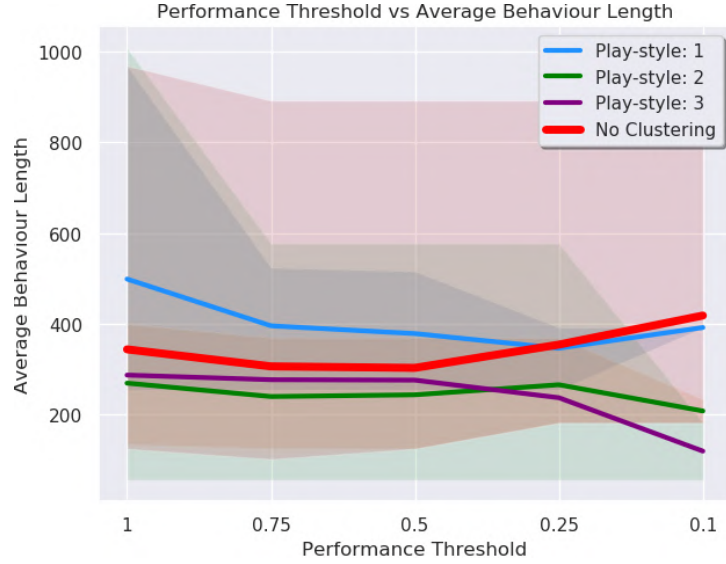


Figure 6.6: Comparison of the effect of differing the performance threshold percentage p on the trajectory diversity in Mario

Finally, Table 6.2 reports the clustering accuracy of each play-style-centric model GM_k with respect to each partition $\mathcal{P}$. The results are averaged over all values of p . The strong performance on the diagonal suggests that each model is specialised in the data within its partition. This finding indicates that the policies learned by each model are inherently different.

Table 6.2: Prediction Accuracy of each model versus the different partitions averaged across all GridWorlds and all p

Average over all values of p	GM_1	GM_2	GM_3	GM_4
$\mathcal{P}_{1,p}$	0.57	0.45	0.44	0.36
$\mathcal{P}_{2,p}$	0.37	0.64	0.36	0.49
$\mathcal{P}_{3,p}$	0.38	0.39	0.63	0.46
$\mathcal{P}_{4,p}$	0.33	0.57	0.40	0.69

6.3.2 Behavioural and skill-based diversity

By varying the value of the performance threshold p we are also able to generate policies of varying skill levels, which is another form of diversity. The reason for this outcome can be attributed to the fact that our PCPG model was trained on an increasing proportion of weaker-performing trajectories. This is evidenced by Figure 6.5, which shows a consistent pattern where increasing values of p correspond to longer average trajectory lengths. This is a valid analogue for skill as the trajectory length was used as the primary feature of our performance metric (Equation 6.1). Furthermore, we can observe that each play-style is converging towards its own optimal length, which serves as further evidence of the behavioural diversity that can be achieved through our PCPG model. In contrast, the “No Clustering” approach converges towards a single behaviour. To further demonstrate diversity across models we observe in Figure 6.7 the cross-correlation between play-style-centric models and the different partitions ($\mathcal{P}_k$). The closer the value is to 1, the stronger the correlation between play-style and trajectory subset. Here it is observed that the model exhibits the highest degree of correlation with the specific subset on which it underwent training. The correlation values that lie outside the diagonal are notably lower, barring a single exception. Specifically, play-styles 1 and 3 exhibit a degree of correlation that is discernible but not significant. This suggests that these two play-styles share some common characteristics, while the remaining play-styles differ significantly from one another.

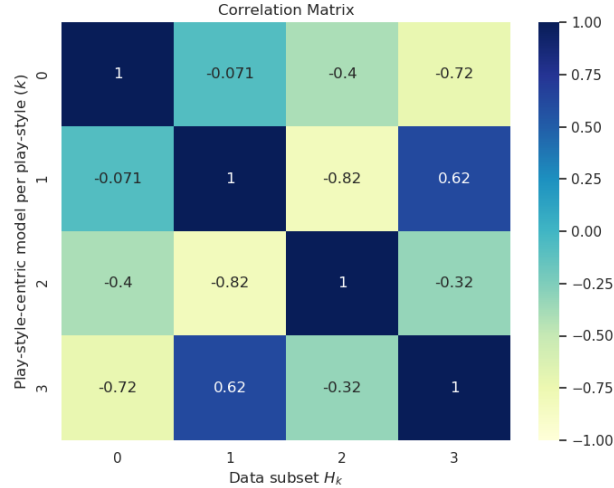


Figure 6.7: Correlation Matrix depicting relationship between play-style-centric models and data subsets across all GridWorld environments ($E_1, \dots, E_5$)

Qualitatively we can view the diversity in behaviours in Figure 6.8 for a particular environment E_2 . Here we observe that the exhibited behaviours match those behaviours encoded in the data, as shown in

Table 4.1. It is also observed that the “No Clustering” behaviour corresponds to the play-style of going directly to the goal, which is expected since these correspond to the shortest trajectories.

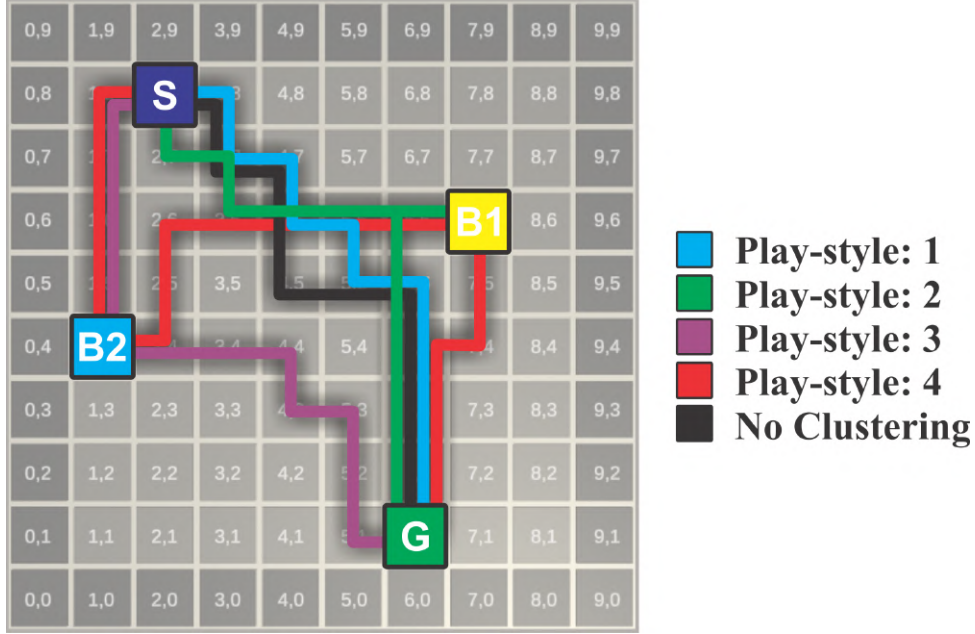


Figure 6.8: Play-style-centric model and “No Clustering” model behaviour visualisation for $p = 0.1$ in E_1

6.3.3 Clustering Performance

Since our approach is dependent on first separating trajectories we look to analyse the effect of the clustering performance of our PCPG model’s performance. In Figure 6.9 we observe the results of manually varying the levels of clustering accuracy. Since our GridWorld data was generated we have access to the ground truth clusters. We then manually mislabel a varying portion of these ground truth labels. Therefore Figure 6.9 is the result of training our play-style-centric models on datasets of varying corruption levels. Here “1” indicates random clustering and “0” is perfect clustering. As a result, we identify that increasing the levels of corruption results in decreasing the effectiveness of our model. However, it does not affect the case where we do not employ any separation. This observation is to be expected as the case whereby we train a single model on the entire dataset does not utilise the play-style information. It is also noted that the variance in performance increases substantially in our models when the level of corruption increases, which indicates that our models are no longer able to specialise in the expected and similar behaviour of the trajectories found in clustered subsets. Additionally, it is worth highlighting that our model’s behaviour remains unaffected by variations in the performance threshold, as demonstrated in Section 6.3.1. In contrast, the model does exhibit sensitivity to changes in the corruption threshold. This contrast implies that our model adeptly clusters individuals exhibiting lower performance levels by their correct play-style, as evidenced by the consistent prediction accuracy despite adjustments to the performance threshold.

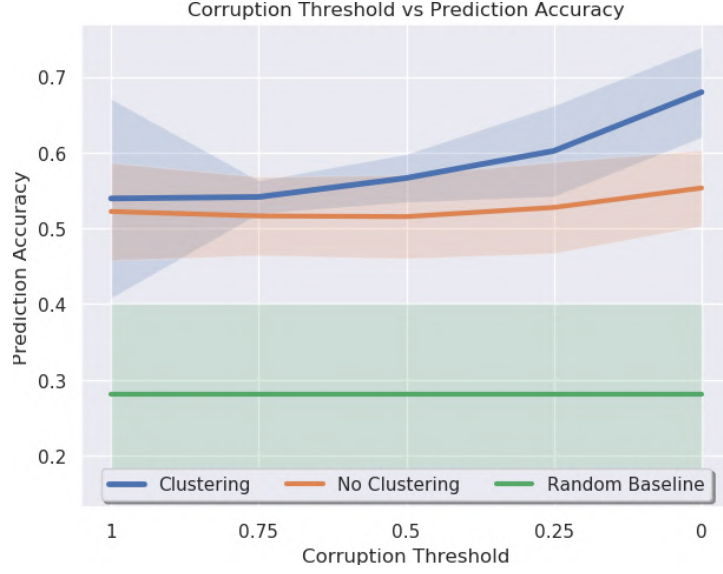


Figure 6.9: Comparison of the effect of differing the corruption threshold percentage on the models’ accuracy. Here “1” indicates random clustering and “0” perfect clustering

The impact of clustering accuracy on the average generated rewards for each play-style-centric model across all GridWorlds is shown in Table 6.3. This form of analysis is only possible as a result of knowing the underlying reward function. Here we observe that perfect clustering results in expected rewards that are identical to those obtained from the original reward functions as defined in Table 4.1 used to generate the GridWorld trajectories. This demonstrates that the play-style-centric models trained on partitioned data can learn to exhibit the underlying play-styles encoded in the trajectories. Additionally, our clustering approach enables us to achieve similar average rewards to the optimal case. However, as clustering performance worsens, the correlation between the underlying reward functions and average obtained rewards degrades.

Table 6.3: The effect of clustering accuracy on the average generated rewards for each play-style-centric model across all GridWorlds

Reward Function	Perfect clustering				Clustering using our model			
	GM_1	GM_2	GM_3	GM_4	GM_1	GM_2	GM_3	GM_4
R_1	99.89	0	0	0	74.07	12.03	13.02	18.24
R_2	0	149.85	0	0	7.42	112.32	11.99	19.22
R_3	0	0	149.86	0	7.42	10.25	115.43	17.38
R_4	0	0	0	199.81	7.30	11.09	16.42	148.52

Reward Function	Clustering with 50% accuracy				Random clustering			
	GM_1	GM_2	GM_3	GM_4	GM_1	GM_2	GM_3	GM_4
R_1	46.87	27.90	24.97	35.56	21.97	39.23	39.85	50.42
R_2	15.32	72.73	29.18	33.27	22.34	36.47	38.09	55.71
R_3	15.65	24.62	76.57	33.59	23.50	36.90	40.80	49.20
R_4	13.27	26.17	25.87	103.88	22.49	38.27	40.60	49.66

6.4 Conclusion

Defining or learning a set of play-style-centric behaviours can be challenging due to the subjective nature of play-styles and the diversity of play-style preferences among players. Additionally, it can be difficult to identify and quantify the defining characteristics of each play-style and create a suitable framework for learning these behaviours. This paper proposes an approach to generating a diverse set of play-style-centric agents that exhibit expected behaviours corresponding to their play-style. Our approach involves utilising unsupervised clustering to separate multi-dimensional varying-length gameplay trajectories by play-style. The resulting trajectory subsets are used to train models using behavioural cloning, enabling the learning of representative policies. This study demonstrates that the separation approach results in models that specialise in the subsets of trajectories on which they are trained and outperform the base case where no separation was used. This paper also demonstrates that model diversity can be achieved through the scaling of a performance threshold, which allows for the learning of agents with varying skill levels. This scalability is useful for developers in creating realistic behaviours that can be more easily scaled in terms of difficulty. Furthermore, this is accomplished without the requirement for designing or learning reward functions, which can be a challenging and non-intuitive process. Notably, this study shows that clustering accuracy has an impact on performance, but decreased clustering accuracy only results in the model prediction accuracy tending towards the baseline. These results were obtained through testing in both synthetic and natural domains, highlighting the method’s robustness and generalisability. In conclusion, our proposed PCPG model, which involves both unsupervised clustering and behavioural cloning, is an effective approach for the creation of a diverse set of play-style-centric policies.

Chapter 7

Player Modeller

The content in this chapter is already published [Ingram *et al.* 2022b] *Branden Ingram, Benjamin Rosman, Clint van Alten, and Richard Klein. Improved action prediction through multiple model processing of player trajectories. In 2022 IEEE Conference on Games (CoG), pages 548–551. IEEE, 2022.*

7.1 Introduction

The task of tailored advice generation depends heavily on the ability to form a representation of the user’s characteristics, preferences, and past behaviours which we call a user (player) model. Particularly, it allows for more effective penalisation of advice or recommendations. By understanding a user’s preferences and behaviours, tailored advice can be generated that is more likely to be relevant and useful to the user.

We look to solve the problem of action prediction given a dataset of trajectories captured from video games. However, video games are dynamic and temporal domains containing long-term dependencies which all affect decision-making, thus making it difficult to train accurate prediction models. In particular, our goal is to train a model which can forecast the behaviour of a particular user rather than an optimal policy. In a competitive setting, this can help players outperform others or from a developer’s perspective predictions can be utilised to modify future gameplay experiences. For our particular use case, we leverage this model to aid in the generation of tailored advice in an online fashion. Common approaches to player modelling have revolved around future action prediction; these include using machine learning techniques such as decision trees [Cheung Chiu and Webb 1998], neural networks [Oh *et al.* 2015], and hidden Markov models [Luckin and others 2007]. These methods can be effective, but they often require large amounts of training data and computational resources.

In addition to generating an accurate model of a user, there is an added requirement of needing to generate this model in a sample-efficient manner. The reason for this is that players’ preferences and behaviours can change rapidly, and advice that is not timely or relevant may not be useful to the player. Furthermore, generating tailored advice quickly can help to improve the player’s overall experience with the game or application. By providing timely and relevant advice, the player is more likely to engage with the game or application and achieve their goals, leading to a more positive experience.

Pretraining is a common technique used for sample-efficient model learning [Zoph *et al.* 2020]. It can be especially useful when the target dataset is small or when labelled data is limited [Yarats *et al.* 2021]. In such cases, training a model from scratch on the target dataset may result in overfitting or poor generalization [Martens and Dardenne 1998]. Pretraining on a large, related dataset can help to

regularise the model and improve its ability to generalise to new data. In the context of tailored advice generation, pretraining can help to capture common patterns and behaviours that are shared among users of similar play-styles, which can improve the accuracy and usefulness of the advice.

Our main contribution is the introduction of a novel neural network-based model, termed Cluster Assisted Prediction (CAP), for predicting future actions. This model improves prediction accuracy by utilising identified play-style clusters to assist in the learning process. By incorporating cluster information, each sub-component of the model only needs to learn from semantically similar data, as opposed to the entirety of the dataset. This enables the sub-components to specialise on their respective data-subsets, leading to enhanced overall performance. The CAP model undergoes initial offline pre-training using a trajectory dataset, followed by online fine-tuning for a particular user to mitigate the delay in accurate predictions as a result of new unseen data.

7.2 Methodology

We aim to solve the following problem: given a user’s partial trajectory ($P = X[1 : t]$) can we accurately and efficiently predict the action which produces the next state in P ? Furthermore, can this approach be utilised in a sample-efficient manner, to accurately predict the actions of new users?

7.2.1 Cluster Assisted Prediction (CAP)

Our proposed CAP model depicted in Figure 7.1 operates in two modes. The first is an offline stage whereby we pre-train the network on a dataset ($\mathcal{D}$) of trajectories. In this process, every trajectory ($X \in \mathcal{D}$) undergoes a gating mechanism and is directed to a specific Predictor Node (M_{k_i}). This ensures that each $X \in \mathcal{D}$ is processed by a single M_{k_i} , which is chosen based on an associated cluster index (k_i). The index corresponding to the appropriate cluster is obtained by utilising the trained Play-style Identification (PI) model, as discussed in Chapter 5. The PI model generates a set of play-style partitions, denoted by $\mathcal{P}_{k_i}$, where $\mathcal{P}_{k_i} \subset \mathcal{D}$ and k_i represents the index associated with a specific play-style partition to which X belongs. Here $k_i \in \{1, \dots, k\}$ where k is the number of identified clusters. The value of k also dictates the number of Predictor Nodes found in our proposed CAP model for a given dataset. These Predictor Nodes, described in further detail in Section 7.2.2, then predict the action required to move from $X[t]$ to $X[t + 1]$ where t represents the current timestep.

To use the player model in an online fashion, we need to ensure that it can adapt to the specific user’s behaviour as they interact with the virtual environment in real-time. Therefore, we employ online training of our CAP model to fine-tune it on the individual user. During the online mode, we continuously train on partial trajectories ($P = X[1, t]$) obtained from a particular user. This training occurs within the same environment that was used to pre-train the CAP model. The pipeline operates similarly to the offline mode, except that we identify the cluster index for the currently available partial trajectory P using the PI model. We then utilise the gating mechanism and direct the partial trajectory to the corresponding predictor node (M_{k_i}), based on the associated cluster index (k_i), to predict the user’s next action. This training occurs in real-time as the user is engaged with the environment. As new data is collected, it is fed into the model to refine its predictions.

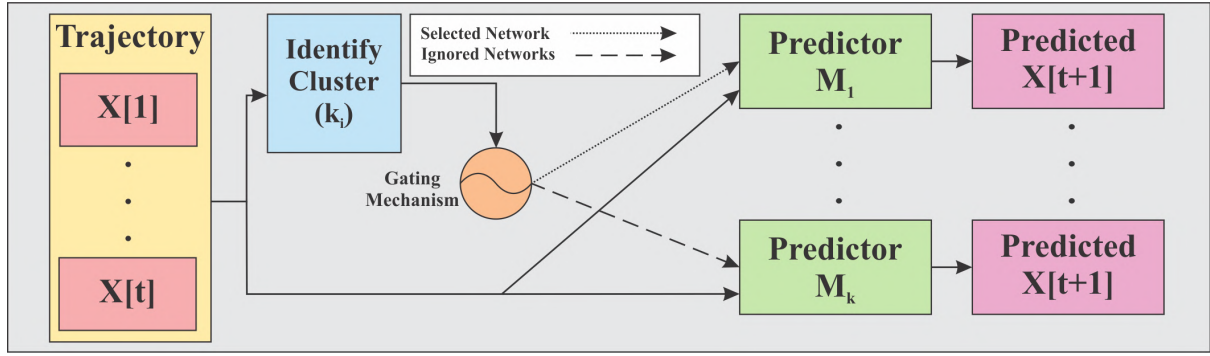


Figure 7.1: Cluster Assisted Prediction Model (CAP)

7.2.2 Predictor Nodes

Our proposed CAP model makes use of a set of Predictor Nodes M_{k_i} , where each node is only trained on data from its corresponding partition $\mathcal{P}_{k_i}$. The individual Predictor Node architecture is depicted in Figure 7.2. The objective for each of these nodes is to forecast the succeeding time step ($X[t + 1]$) of a given partial trajectory ($P = X[1 : t]$) present within the trajectory partition ($\mathcal{P}_{k_i}$), whereby t denotes the timestep. Our Predictor Nodes are comprised of two components. The first component is a standard LSTM layer which is required since P varies in length as we receive more input from a user. This layer accepts as input any partial trajectory and outputs a one-dimensional vector which represents an encoding of the trajectory. This encoding is then appended to the current time step ($X[t]$) of the partial trajectory and serves as the input to the second component, our fully connected neural network. Finally, this node outputs the next state as its prediction ($X[t + 1]$).

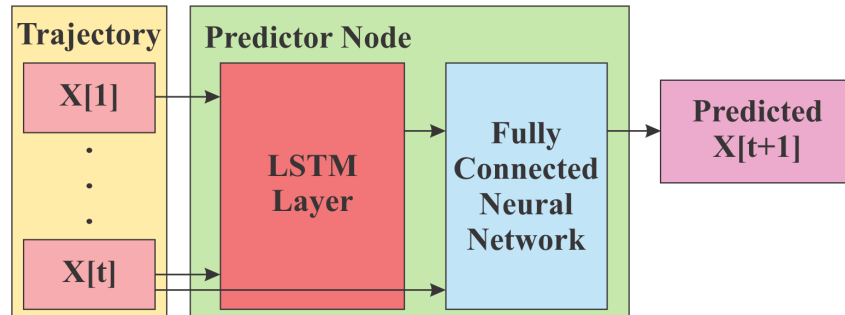


Figure 7.2: Individual LSTM Predictor Node

7.3 Offline Pre-Training

We trained our CAP model on a set of trajectories ($\mathcal{D}$) where the cluster for each $X \in \mathcal{D}$ was identified using the PI detailed in Chapter 5 and is represented as k_i . This identification is important as it dictates which Predictor Node our input trajectory should be passed to. The fully connected neural network component found in each Predictor Node of our CAP model consists of 3 hidden layers of 5 neurons with ReLu activations. Additionally, these nodes utilised a non-stacked LSTM which outputs an encoded state of size 8. By appending this encoded state by the last time step of the partial trajectory we get the input of the fully connected neural network. The input dimension of both the fully connected component and the LSTM layer is, therefore, dependent on the dataset's timestep dimension. For comparison, we trained a model (Single) with the same architecture as depicted in Figure 7.2 on the complete dataset ($\mathcal{D}$). This model has the same structure as a single Predictor Node found in our CAP model. The difference

is the size of the hidden layers found in the fully connected component. To ensure fairness between our CAP and Single models, we keep the total capacities of the models roughly equal. This is achieved by increasing the size of the hidden layers in the Single model by a factor dependent on the number of clusters. For the Single model, we used a hidden layer size of $5 \times k$ where k was the number of clusters. The resultant accuracy from this model would serve as the base of comparison with our CAP model.

Both our CAP and Single models were trained for 10000 episodes using the Adam optimiser with a learning rate 0.001 using mean square error as our loss function. During each episode, progressively longer partial trajectories are fed forward through the complete network until the entire trajectory has been processed. For each of these partial trajectories ($X[1 : t]$) the output (y') is compared to $X[t + 1]$ with the loss being backpropagated through the entire network. This process is outlined in Algorithm 9.

Algorithm 9 Player Modeller (PM) Training

Input: Set of trajectories D , CAP model

Output: Trained CAP model

- 1: **for** each trajectory $X \in D$ **do**
 - 2: **for** each partial trajectory $P \in X$ **do**
 - 3: Identify the cluster k_i for P using PI detailed in Chapter 5
 - 4: Select Predictor Node (M_{k_i}) using k_i
 - 5: Encode P using LSTM layer in M_{k_i} to obtain Z_P
 - 6: $Z_P = Z_P \oplus P_t$ where p_t is the final state in the P , and $\oplus$ represents concatenation
 - 7: Pass Z_P through the fully connected NN layer to obtain predicted X'_{t+1}
 - 8: Calculate loss between X_{t+1} and X'_{t+1} using Equation 2.11
 - 9: Back-propagate using loss
-

The efficacy of our methodology relies on the utilisation of grouped data, and as such, is contingent on the accuracy of the clustering process. To showcase a standard of excellence for our approach, we leverage the preexisting knowledge that the grid world data was designed to showcase distinct patterns. This implies that a definitive clustering solution exists. Consequently, we proceeded to train our CAP model on the distinct partitions ($\mathcal{P} \subseteq \mathcal{D}$) that were separated using these ground truth labels. We performed this procedure for all six of our environments ($E_1, \dots, E_5$, Mario). All experiments were conducted using 4 seeds across the 5 GridWorld environments as well as Mario with the average accuracy for each being recorded. The initial dataset ($\mathcal{D}$) contains 8000 trajectories for each GridWorld environment and 74 for Mario. These datasets were then split 70-30 for the training and testing sets.

7.4 Online Fine-Tuning

To train our model for online learning, we use the latest partial trajectories collected from the user to continually train the pre-trained CAP model. To evaluate the effectiveness of our pre-training method, we created a small subset of trajectories, consisting of 5 previously unseen trajectories for each play-style (k), which represents our user-set, denoted as $\mathcal{U}_k$. These unseen trajectories were randomly selected from the testing set. We trained both the pre-trained CAP model and the CAP model without pre-training on $\mathcal{U}_k$ to demonstrate the benefits of pre-training in achieving sample efficiency. Both models are trained for 5 episodes, where each episode involves progressively increasing the amounts of data sampled from $\mathcal{U}_k$. The intra-episode training process is similar to Algorithm 9 described in the offline training process, except that now our training dataset $\mathcal{D}$ is a subset of $\mathcal{U}_k$ that grows by one trajectory until it is equivalent to $\mathcal{U}_k$. This process is repeated for each play-style k .

7.5 Results

For evaluation purposes, we use unseen trajectories from each environment. For each trajectory (X) we process every partial trajectory ($P = X[1 : t]$) through each of the evaluated models to generate predictions y' . The overall performance is the percentage of correct predictions across the given testing set.

7.5.1 Offline Training Model Prediction Accuracy

The results of evaluating the prediction accuracy of the pre-training step are depicted in Table 7.1 with bold values representing the best model for a given environment. Here we observe for each of the environments, whether structurally similar in the case of $E_1, \dots, E_5$ or more complex (Mario), our CAP model outperforms the Single model. Notably, this performance level was achieved in both synthetic and natural domains. This indicates our approach of utilising clustered partitions of data can be beneficial when using real-world data. Since our CAP model is partially dependent on the clustering accuracy, we analysed the effect of using the ground truth clusters available from the GridWorld environments. The results of using ground truth clustering serve as a benchmark comparison to minimise the impact of clustering performance. We observed that our CAP model performed at an even higher level when using the data separated using the ground truths. This indicates that with better clustering our CAP model can achieve greater accuracy.

Table 7.1: Prediction Accuracy on Clustered Data

E	Random	Single	CAP	CAP (ground-truth)
E_1	0.27	0.56	0.6	0.63
E_2	0.28	0.51	0.53	0.6
E_3	0.27	0.34	0.47	0.48
E_4	0.29	0.58	0.62	0.68
E_5	0.27	0.52	0.56	0.57
Average across GridWorlds	0.28	0.50	0.56	0.59
Mario	0.025	0.30	0.36	N/A

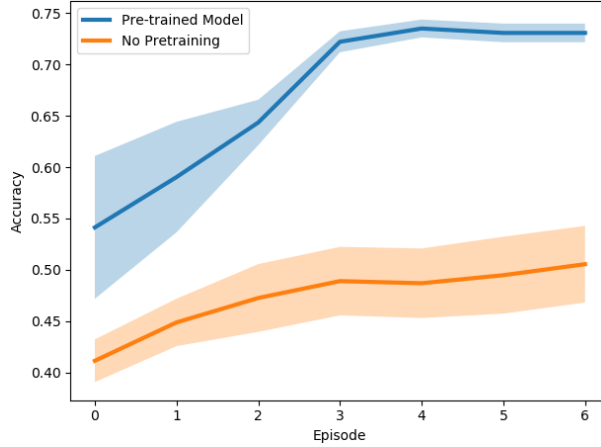
Lastly, we analysed the ability of the individual sub-models ($M_1, \dots, M_k$) to specialise in their respective data partition ($\mathcal{P}_1, \dots, \mathcal{P}_k$) averaged across all environments. This is achieved by comparing the prediction accuracy of each sub-model on all data subsets for a given environment. The results of this process were then averaged across all environments and are depicted in Table 7.2. Here we observe that the prediction accuracy of $M_1 \dots M_4$ is higher when making predictions from trajectories within their corresponding data partition depicted by the bold values. Conversely, the prediction accuracy on trajectories from other partitions is far lower than even the average performance of the overall CAP model depicted in Table 7.1. It is by specialising on these partitions that when combined into our CAP model we obtain the overall benefit. Lastly, we observe that the average prediction accuracy of our CAP model across all environments is greater than the average performance for each sub-model.

Table 7.2: Prediction Accuracy of sub-models on each data subset

	M_1	M_2	M_3	M_4	CAP
$\mathcal{P}_1$	0.65	0.29	0.21	0.2	—
$\mathcal{P}_2$	0.25	0.56	0.19	0.21	—
$\mathcal{P}_3$	0.15	0.15	0.66	0.23	—
$\mathcal{P}_4$	0.22	0.21	0.2	0.44	—
Average	0.32	0.3	0.32	0.27	0.56

7.5.2 Pre-training Effectiveness For Online Training

In Figure 7.3, it can be observed that the pre-trained CAP model outperforms the CAP variant that did not undergo pre-training. Furthermore, our pre-trained CAP model exhibits the ability to quickly specialise on user data, as it demonstrates good performance from the outset. Additionally, the rate at which it specialises on individuals is faster than that of the CAP model without pre-training.

**Figure 7.3:** Prediction Accuracy comparison between pre-trained CAP model versus no pre-training

7.6 Conclusion

This chapter proposes a new multi-model approach dubbed CAP which demonstrates the benefit of pre-training multiple models on separated data over a single model on a complete dataset. Our CAP model was trained to perform optimally in a future action prediction task on multi-dimensional variable-length gameplay trajectories which had been separated with respect to a clustering process. We tested this on a synthetic dataset where cluster information was known as well as a natural domain (Mario) where the ground truth clusters were not known. Through empirical analysis, we demonstrated that our CAP model consistently outperformed the Single model trained on the overall dataset. We also demonstrated that the performance increase came from the ability of the sub-models to specialise on their respective data subsets. We conclude that with the use of an initial data separation process, you can obtain superior performance when training future state prediction models in video game environments. Finally, we have shown that by leveraging the pre-training process, where our CAP model was trained on an existing dataset of trajectories, we were able to develop a model that could swiftly adapt to previously unseen trajectories from a specific user. This result serves as evidence of the model’s efficacy in online scenarios.

Chapter 8

Advisor

8.1 Introduction

Learning how to advise in an efficient and engaging way has long since been a problem in numerous domains including teaching [Feiman-Nemser 1983], robotics [Kober *et al.* 2013] and video games [Shaffer *et al.* 2005]. Doing so in an automatic fashion which is tailored to a specific user is an even more challenging problem. Specifically in video games, this involves generating advice that is helpful, relevant, and actionable for individual players that require overcoming obstacles such as personalisation and human-like reasoning [Brisson *et al.* 2012]. Generating tailored advice in video games is important for learning and skill development, enjoyment and engagement, which can enhance the player experience and contribute to a more personal, enjoyable, and engaging gaming experience.

Machine learning [Char *et al.* 2018], natural language processing [Papamichail and French 2003], and expert systems [Ye and Johnson 1995] have been utilised for the generation of advice for recommendation systems. These models analyse a user's data and provide personalised feedback and advice based on the specific context and goals of the user. Overall, while recommender systems can be useful for making personalised recommendations, they may not be sufficient for learning to be better. Specifically in terms of games, many also incorporate tutorial systems or walkthroughs that provide advice and guidance to players as they progress through the game [Andersen *et al.* 2012]. However, these systems tend to exhibit a limited scope and lack of personalisation due to their one-size-fits-all nature. Thus, generating tailored advice that is helpful, relevant, and actionable remains a challenging problem that requires overcoming several obstacles, including the need for personalisation, domain expertise, and human-like reasoning. Automated systems may not account for individual differences in play-style, preferences, and goals. Therefore, new approaches such as our model comparison approach discussed in this chapter are needed to augment existing automatic advice generation systems and improve their effectiveness.

Comparing models of individuals with that of an expert, such as a teacher or mentor, is a promising approach for garnering tailored advice and improving learning rates [Bullough Jr *et al.* 2003]. This approach involves creating a model or profile of the novice player's gameplay and comparing it to that of an expert. By analysing the differences between the two models, the system can identify areas where the novice player could improve, and offer tailored advice to help them progress. The expert's behaviour can be used to augment the automated system, providing a more comprehensive understanding of the game mechanics and strategies. This approach can be seen as a form of apprenticeship learning, where novice learners receive targeted instruction from experts and progress through a structured learning pathway [Abbeel and Ng 2004]. The expert's behaviour can be used to create a curriculum of tasks or challenges that build upon each other, guiding the novice player towards mastery. Additionally, transfer

learning techniques can be used to adapt the expert’s knowledge to the novice player’s context, such as by identifying commonalities between different situations [Torrey and Shavlik 2010].

In this chapter, the ultimate component of our automated advice generation system is presented, namely the Advisor (AD). The chapter proposes the integration of the previous three components, namely Play-style Identifier (PI 5), Game Modeller (GM 6), and Player Modeller (PM 7), into an operational online format. This process is described in Section 8.2. In particular, the PI provides a way to identify a player’s unique play-style, which can inform the selection of the expert model to be used for comparison. Through this approach, we aim to teach a player to be better with respect to their identified play-style, which can be obtained from the PI. In Section 8.3, we delineate our experimental framework that was employed to produce the encouraging and exploratory findings presented in Section 8.4. The success of open-ended games like Minecraft¹ which allow players the freedom to play in their own style as well as their application in learning [Nebel *et al.* 2016] leads us to believe that approach can lead to increased user engagement, motivation, and enjoyment. Additionally, the PM provides an accurate model of a novice learner allowing for the understanding of their motivations, behaviours and preferences, while the GM provides the expert model which is aligned to those needs. By comparing the two models, the system can identify differences between novice and expert gameplay and offer personalised advice to help the player progress. The overall goal is to minimise the disparity between the user and their corresponding expert.

8.2 Methodology

Our methodology is designed such that it is capable of being used in real-time while a user is playing, wherein it offers guidance to the user during their interaction with a video game. The interaction between a player and a game environment can be described as a cycle as seen in Figure 8.1, consisting of four stages:

- **Input:** The player inputs commands or actions into the game, such as moving a character or selecting an item.
- **Processing:** The game processes the player’s input and updates the game state accordingly, such as moving the character to a new location or updating the inventory.
- **Output:** The game outputs the updated game state to the player, such as displaying the new location of the character or the updated inventory.
- **Feedback:** The player receives feedback from the game, such as a reward for completing a task or a penalty for failing to complete a task. This feedback can influence the player’s future input and behaviour in the game.

¹www.minecraft.net/

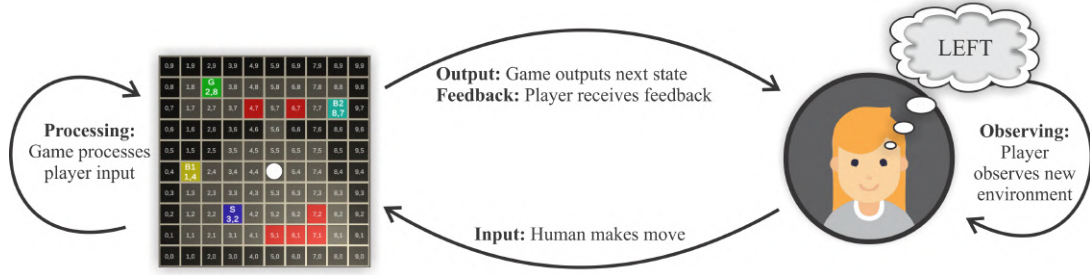


Figure 8.1: Cycle of interaction between user (player) and environment (video game)

It is during this feedback stage that our model augments standard feedback to include tailored advice. Given this consideration, we aim to tackle the following problem: can the utilisation of identified play-style information (pk) to train precise models of a user (pm) and their expert self (gm_{pk}) facilitate the reduction of the disparity between them?

8.2.1 Advice Generation

This overall process of advice generation in an online setting is outlined in Algorithm 10. We associate each user ($\mathcal{U}$) with a parameter ϖ which serves as a measurement of the likelihood of the user accepting advice (Line 1). By introducing this thresholding parameter, the model is able to simulate the practical scenario of users rejecting advice, thereby enhancing the realism of the system.

Algorithm 10 Advice-giving Process

```

1: procedure ADVICE-GIVING( $\mathcal{U}, \varpi$ )
2:   Initialise  $pi$ ,  $pm$  and  $gm$  models
3:   Initialise Memory
4:    $P \leftarrow X[1 : t]$  ▷ This is the trajectory observed we observe from  $\mathcal{U}$ 
5:    $pk \leftarrow pi(P)$  ▷ using Play-style Identifier obtain play-style  $pk$  from  $P$ 
6:    $\pi_{pk}^* \leftarrow gm(pk)$  ▷ using Game Modeller obtain optimal play-style-centric  $\pi_{pk}^*$  from  $pk$ 
7:    $pm \leftarrow pm.train(P)$  ▷ Fine-tune player model with new user data
8:    $a_e \leftarrow \pi_{pk}^*(P)$  ▷ Predict optimal action using  $\pi_{pk}^*$ 
9:   if  $P \in Memory$  then ▷ If previously advised on current partial trajectory
10:     $a_p \leftarrow Memory[P]$  ▷ Use advised action
11:   else
12:     $a_p \leftarrow pm(a_p)$  ▷ Predict player action using  $pm(a_p)$ 
13:   if  $a_e \neq a_p$  then ▷ Equation 8.1
14:     Inform the user of the disparity
15:      $r \in [0, 1)$ 
16:     if  $r \leq \varpi$  then ▷ Check to see if user takes advice
17:        $Memory[P] \leftarrow a_e$  ▷ Add expert action into memory for current  $P$ 

```

At the outset of a user’s interaction with the system, we initialise the three models that were previously introduced in Chapters 5, 6, and 7, namely the Play-style Identifier (pi), Game Modeller (gm), and Player Model (pm), respectively (Line 2). In addition, a memory buffer is created to keep track of instances when the user accepts advice. The buffer stores the current partial trajectory P , representing the current sequence of states the user has visited, and the corresponding “correct action” provided by the expert. In continuous domains function approximation can be used to generalise experiences, allowing

the model to learn from a smaller set of experiences [Lillicrap *et al.* 2015]. The subsequent step involves determining whether advice is required, which entails predicting the next action of the player and their corresponding expert, taking into account their play-style. This prediction is made based on the user's current partial trajectory P .

The process of determining the action of the player involves several steps. Initially, the memory buffer is queried (Line 9) to verify whether the user has received advice for the current partial trajectory (P) in the past. If the memory buffer contains P , the advised action that was previously stored in the memory is selected as the user's upcoming action a_p (Line 10). In this way, we assume that if a user had previously decided to remember the given advice then the user will act on it if possible. On the other hand, if P is not present in the memory buffer, the player model pm predicts the user's action a_p (Line 12). Furthermore, we fine-tune the pm by training the model on the observed partial trajectories of the user (Line 7). The key goal of the pm is to achieve a precise representation of the user's actions by focusing on the latest incoming user-specific information during system usage. This objective is accomplished by continuously learning from the observed partial trajectories P as the user continues to interact with the system, as demonstrated in Figure 8.2. As we receive additional data from the user, the model can specialise further on that individual, leading to more reliable predictions.

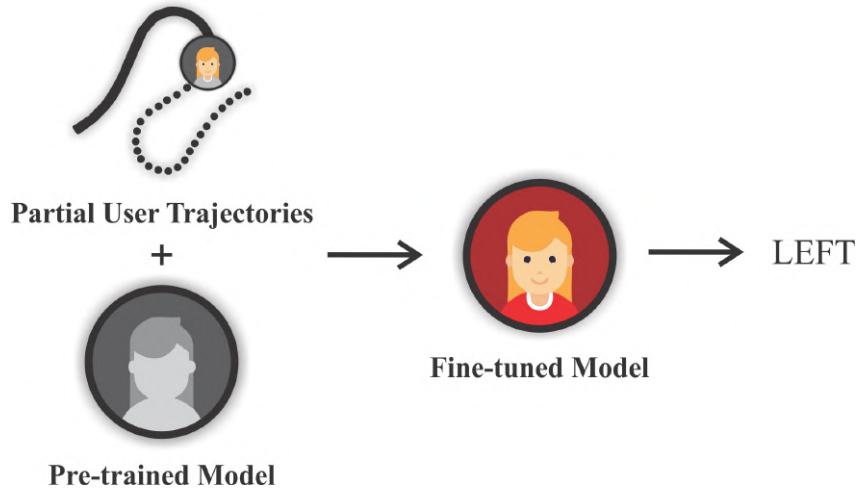


Figure 8.2: Online functioning of the player model (PM). Upon initial usage of the system, a pre-trained model is utilized as the player's model. However, as additional information regarding the user's gameplay is obtained through partial trajectories, the model is updated to enhance its ability to accurately predict the user's behaviour.

In addition to acquiring the user's action (a_p), we also determine the optimal action a_e based on the partial trajectory P . This involves the identification of the user's expert play-style policy (gm_{pk}), which is continuously obtained from the set of policies learned by the GM, as explained in Chapter 6 (Game Modeller). This identification process consists of several steps, as shown in Figure 8.3. During the user's interaction with the game, the system records information on their gameplay in the form of a partial trajectory (P). The trajectory is then encoded (Z_P) using the trained PI model, as detailed in Chapter 5 (Play-style Identifier), using the $Encoder(P)$ method as presented in Equation 5.1. Next, the user's play-style (pk) is determined by minimising the average distance between the cluster centroid and the latent encoding (Z_P), as specified in Equation 5.4, i.e., $pk = \operatorname{argmin}(\bar{d}(Z_P))$. The play-style identifier (pk) refers to the index of the play-style cluster centroid that is closest to the latent encoding (Z_P). Ultimately, the expert play-style-centric model (gm_{pk}) corresponding to the cluster identifier pk is selected (Line 5). Finally, the optimal action (a_e) for the specific user is determined using the policy (π_{pk}^*) of the expert play-style, given the partial trajectory P (Line 6).

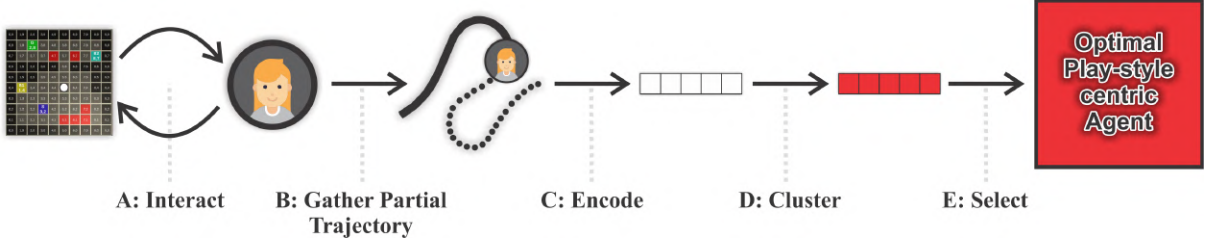


Figure 8.3: Five-step process of identifying an expert play-style-centric version of the current user. Firstly, the user interacts with the video game (A). Secondly, the system collects the user’s current trajectory (B). Thirdly, this partial trajectory is encoded online (C). Fourthly, the encoding is clustered (D). Finally, the corresponding play-style-centric policy is selected using the cluster identifier (E).

These user-specific models (pm and gm_{pk}) are then compared (Line 12) on a per action basis to identify differences using Equation 8.1. This function computes the action vectors outputted by the two models given the partial trajectory and then computes the Euclidean distance between them.

$$\text{distance}(pm, gm_{pk}, P[1:t], t) = \sqrt{\sum_{i=1}^n (pm(P[1:t])_i - gm_{pk}(P[1:t])_i)^2} \quad (8.1)$$

Figure 8.4 illustrates a sequence of comparisons between the user model (depicted on the left) and the corresponding expert play-style-centric model (on the right). In the first cases, the user’s actions match those of the expert, and hence, no advice is given. However, in the third case, a discrepancy is observed between the user’s actions and the expert’s, indicating that advice should be sent to the user. Upon receiving advice we consider the scenario where the user can decide whether to remember the advice or not. If the user chooses to accept the advice, based on the parameter ϖ , the current partial trajectory is stored in the memory for future reference.

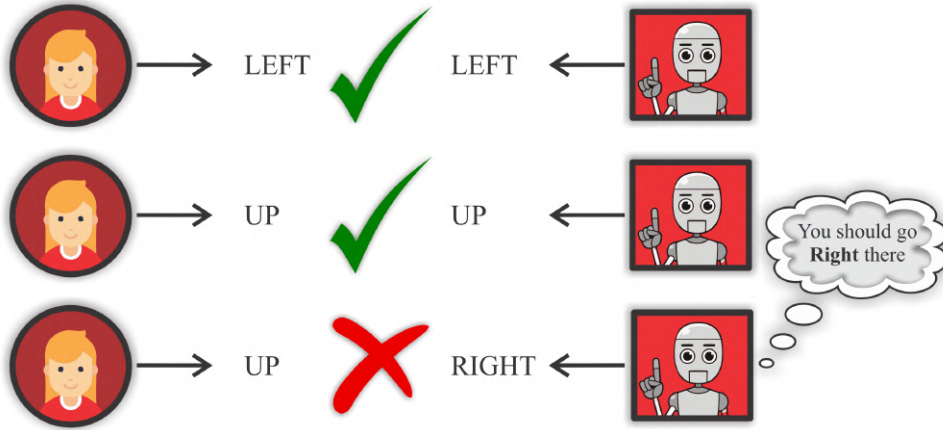


Figure 8.4: Example comparison between player model and corresponding expert play-style-centric model

8.3 Experiments

This section describes the experimental protocol developed to assess the effectiveness of the entire system in improving user performance using Algorithm 10. The experimental procedure involved constructing a dataset of individual users by randomly selecting five trajectories from a larger dataset ($\mathcal{D}$)

from the GridWorld domain, which is referred to as a user-set ($\bar{U}$). For diversity, four selection schemes (outlined in Table 8.1) were used to select the trajectories, each representing a different user archetype. The term “skill” in this context denotes the level of proficiency of a user, independent of their playing style, assessed by the same metric outlined in Equation 6.1 as explained in Chapter 6. The term “style” represents the user’s play-style irrespective of their performance. Therefore, our four user archetypes are represented by the different combinations of these two features.

Table 8.1: Different user-set trajectory selection schemes with their associated description

Selection Scheme Name (Archetype)	Description (Trajectories are randomly selected from entire dataset $\mathcal{D}$)
full-random	Trajectories do not share play-styles or skill level
random-skill	Trajectories have different play-styles and relatively similar skill levels
random-style	Trajectories share the same play-style but with varying skill levels
set-skill-and-style	Trajectories share the same play-style and relatively similar skill levels

The aforementioned process (Algorithm 10) was repeated for a total of eight users, for each selection scheme and for each value of ϖ within the range of $[0, 0.25, 0.5, 0.75, 1]$. A value of “0” indicates a user who never accepts advice, while a value of “1” denotes a user who always accepts advice. Additionally, an infinite memory buffer was used, which means that once a user accepted advice, it would never be forgotten. The correlation score between each user and their corresponding expert was calculated and recorded as the performance metric for our model. This metric is chosen as the objective of our model is to minimise the disparity between a user and their identified expert. Therefore the approach of minimising the distance between our expert and the user is a valid approach as we have previously demonstrated in Chapter 6 that the play-style-centric models, when trained on high-performing trajectories, result in high-performing policies with respect to the play-style’s underlying reward function.

8.4 Results

The experimental results regarding the impact of the tailored advice generation pipeline on various advice acceptance thresholds and user archetypes are presented in Figure 8.5. As expected, it is observed that when advice is never utilised, the advice generation model fails to assist the user in improving their performance. However, in every other case where even marginal levels of advice utilisation occur, the model performance across all archetypes improves. Furthermore, as the degree of utilisation rises, there is a corresponding increase in the proportion of performance improvement.

Notably, it is observed that the “set-skill-and-style” archetype outperforms all others except in the “never take advice” scenario. Nevertheless, improvements were seen across all archetypes, indicating that the system provides tailored advice. The “set-skill-and-style” archetype performs best because the model is best able to exploit the consistent play-style characteristics found in this user-set. Similarly, the model performs the second best in the “random-skill” archetype as this archetype also has a consistent play-style. Both the Game Modeller and Player Modeller are trained in play-style-centric fashions rather than skill-centric, which explains these results. Lastly, it is worth noting that having a consistent skill level is more conducive to improvement compared to randomness.

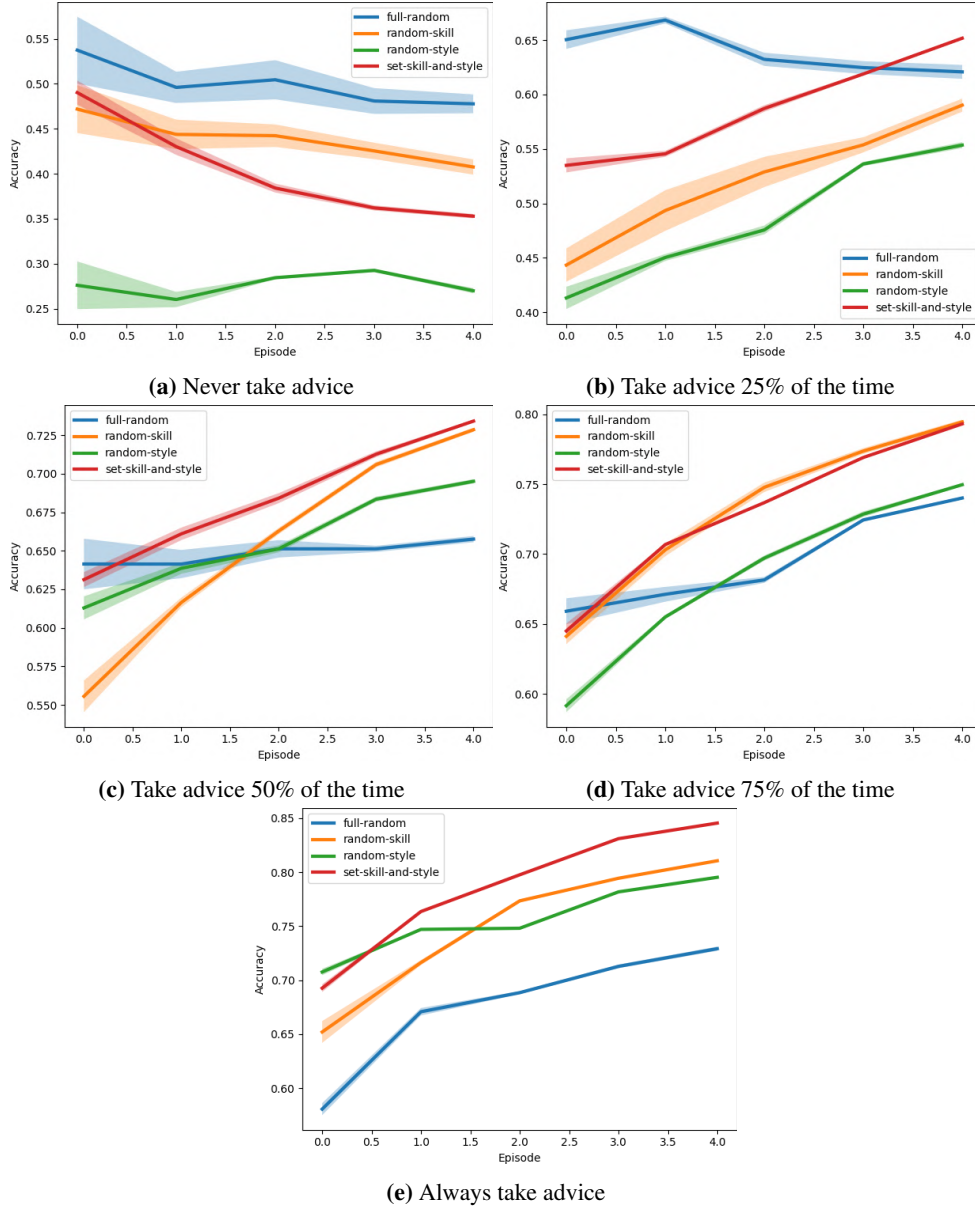


Figure 8.5: Comparison of the impact of a tailored advice generation pipeline on various advice acceptance thresholds and user archetypes for GridWorlds domain.

We employed the identical experimental configuration within the MiniDungeons domain, and the outcomes are depicted in Figure 8.6. In this context, it is evident that, similar to our previous observations, when the advice is consistently disregarded, our model does not contribute significantly to improving the performance of the simulated user. Furthermore, as we observed in the GridWorlds domain, allowing the model’s advice to be followed tends to lead to performance enhancements in our user model. However, this phenomenon is less pronounced in the MiniDungeons domain, particularly for the “full-random” and “random-style” user archetypes. Additionally, we observe slower performance improvements, which can be attributed to the increased complexity of the MiniDungeon trajectories, requiring users to accumulate more experiences before entering states they have encountered previously. Notably, the most effective user archetypes remain “set-skill-and-style” and “random-skill”, both of which correspond to users with consistent play styles. This indicates that our play-style identification model can

better comprehend our users, enabling the generation of more tailored advice, ultimately leading to a more substantial impact on their ability to emulate their respective experts.

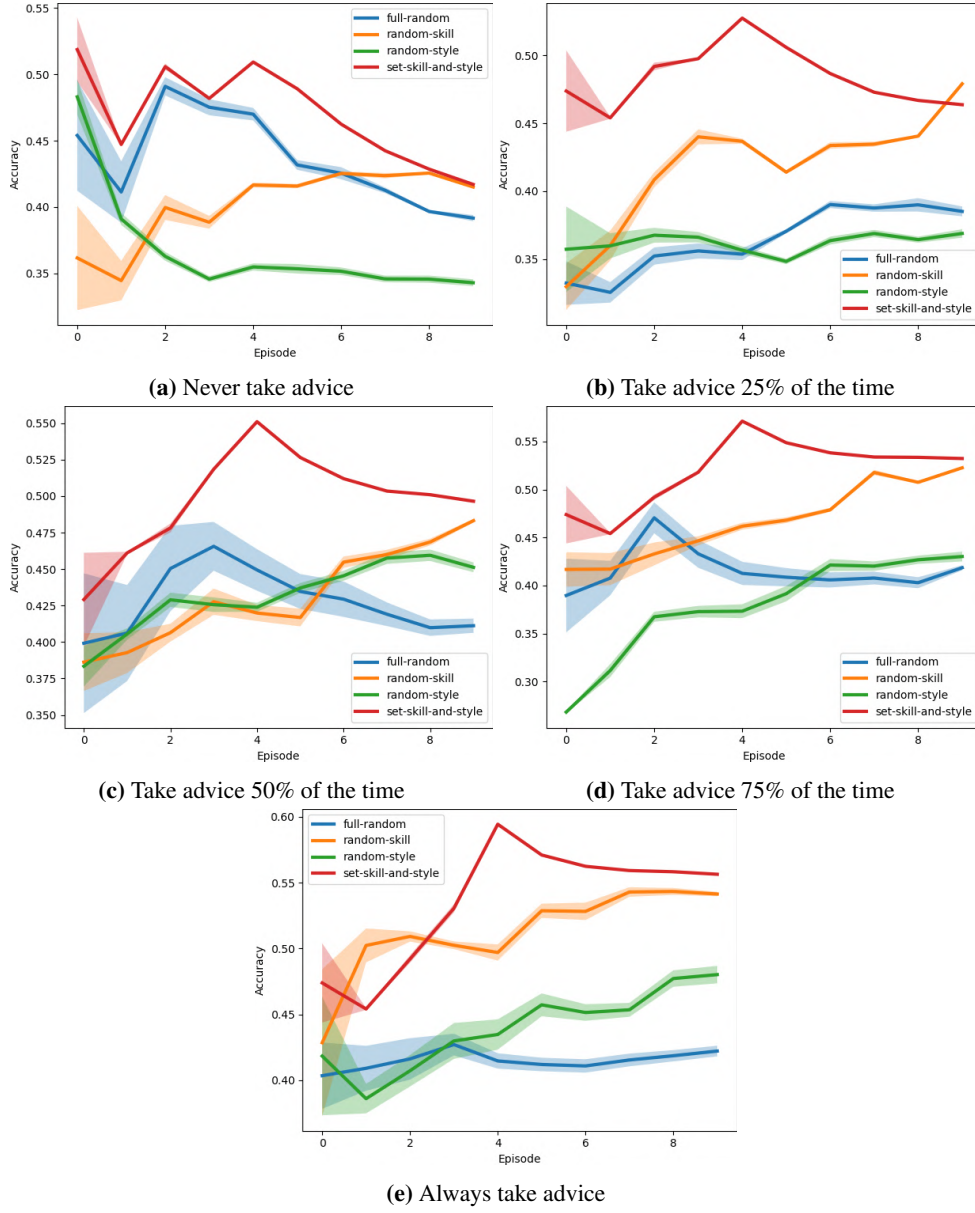


Figure 8.6: Comparison of the impact of a tailored advice generation pipeline on various advice acceptance thresholds and user archetypes for MiniDungeons domain.

8.5 Conclusion

In this chapter, we have presented the Advisor (AD) component of our automated advice generation system which integrates the previously developed PI, GM, and PM into an operational online format. Specifically, our aim was to provide an illustration of how these interconnected components could be used to generate personalised advice for individual users through the use of a modal comparison technique. Our experimental simulations have demonstrated promising results in achieving the desired goal of minimising the disparity between users and their corresponding experts.

Furthermore, we have shown that the PM and GM models can accurately identify differences between novice and expert gameplay, which allows for the provision of personalised advice to help players progress. While additional research is necessary to fully investigate the potential of our system, we hold a strong belief that our efforts constitute a noteworthy advancement towards creating efficient automated advice generation systems that can be utilised in diverse applications.

In conclusion, our AD component has demonstrated the ability to provide tailored advice to individual users based on their gameplay characteristics, leading to improvements in performance. By integrating this component with our previously developed PI, GM, and PM, we have created a comprehensive automated advice generation system that has the potential to benefit a range of applications, including gaming and education.

Chapter 9

Conclusion

Our aim for this thesis was to be able to create a system for the automatic generation of beneficial and tailored advice through the utilisation of identified play-style information. In the context of our initial example, our approach involved identifying a player's style, such as defensive, as a primary step to furnish recommendations that could assist them in achieving the pinnacle of their defensive capabilities.

Therefore, the main contribution of this thesis aimed to develop an end-to-end system capable of automatically generating tailored advice for video game players. To this end, we proposed an overall advice generation model as seen in Figure 9.1 that was made up of four separate interconnected components that were trained in both online and offline settings. These individual component's namely; the Play-style Identifier (PI), the Game Modeller (GM), the Player Modeller (PM), and the Advisor (AD), although interconnected, each looked to answer a separate question.

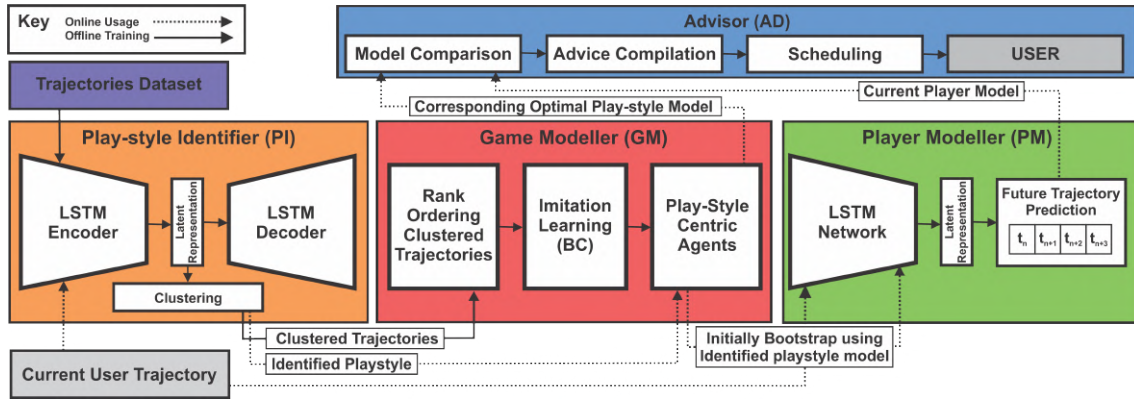


Figure 9.1: Complete Pipeline of Play-style-centric Advice Generation

Firstly, we demonstrated how our proposed PI was able to identify, analyse, and interpret different play-styles through an unsupervised clustering autoencoder approach. Here we demonstrated the ability of the model to identify play-styles of users in an online fashion from partial trajectories without the need for further training. Additionally, through the analysis process, we were able to discern not only the characteristics of individual play-styles but also their relation to others, which was something previously unstudied in video games.

Secondly, we demonstrated how our GM was capable of learning policies representative of the best way to play for each identified play-style. This was achieved through a novel clustering-assisted behavioural cloning model dubbed PCPG which learns play-style-centric policies in a supervised fashion

from demonstrations separated with respect to play-style. Here we demonstrated that not only is this a viable technique for learning accurate policies representative of expected behaviours but is also an efficient technique for learning a diverse set of policies in terms of both skill and behavioural diversity.

Thirdly, we showed how the PM was able to model the play-style of individual users. This was also achieved through a cluster-assisted model dubbed CAP. We demonstrated that this model was able to accurately predict user action necessary for forecasting the future states of a user by leveraging the identified play-style. Furthermore, we demonstrated that through the use of a pretraining procedure, we were able to limit the lag in performance essential for using such a model in an online setting.

Lastly, we discussed how our AD was able to utilise each of these previous components to produce advice specific to the particular user. This was demonstrated through the application of a model comparison approach whereby the player’s model was compared with their corresponding optimal play-style model to formulate useful advice.

9.1 Limitations

In this section, we outline the two main limitations of the presented work. These are the misclassification rate of the different models as well as the lack of exploration in complex domains. We discuss potential solutions to these issues in Section [9.2](#)

9.1.1 Misclassification Rate

One of the primary limitations of our model’s play-style identification task is its misclassification rate. While achieving roughly 70% accuracy is a significant accomplishment, the remaining 30% misclassification rate cannot be overlooked. This level of inaccuracy has the potential to undermine the utility and credibility of the model’s predictions. Individuals who are misclassified might receive recommendations, personalized experiences, or interventions that do not align with their actual play-style, leading to frustration, disengagement, and a lack of trust in the model’s capabilities. One key factor contributing to this misclassification is the similarity in the starting points of trajectories. Play-styles tend to diverge and become more distinct as the game or activity progresses. Players with different play-styles will eventually make different decisions, adopt unique strategies, and exhibit varied behaviours. However, during the initial phases, these differences might not be apparent, leading to misclassification. This behaviour can be observed in Figure [5.14](#) where the model initially fluctuates between multiple play-styles before converging.

9.1.2 Continuous Action and State Space Domains

Another notable limitation of our work is the lack of testing in continuous action and state space domains. The model’s performance might differ when applied to scenarios that involve continuous and dynamic interactions, such as real-time strategy games, sports simulations, or virtual reality environments. In discrete tasks, the model’s ability to classify play-styles might be more straightforward due to the distinct categories. However, continuous domains introduce large state and action-based complexities. Failing to adequately test the model’s capabilities in such contexts might result in suboptimal performance or even inaccurate insights. Addressing this limitation would require extending the model’s training and evaluation to encompass a wider variety of scenarios that better reflect the continuous nature of certain activities and games. However, the overall model’s architecture as presented in Figure [9.1](#) can still be applied to these domains with minor alterations. These alterations would involve changing representations such that our model could process the data. For example, this could involve incorporating convolutional layers into our networks to process image-based trajectories.

9.2 Future Work

In this section, we outline several potential directions for improving our model for generating automated tailored advice in video games. Given that our overall model for automated tailored advice generation in video games consists of multiple components, each of which serves a specific purpose, it is possible to make individual improvements to each of these components as well as to explore how they interact with one another.

9.2.1 Play-style Identifier (PI)

One potential avenue for future research with the PI component of our model would be to explore its application on image-based trajectories. This could be captured from the screen recordings of individuals while they were playing a video game. Convolutional autoencoders have shown effectiveness in image segmentation and classification tasks [Guo *et al.* 2017] and could be a suitable architecture for clustering sequences of screen frames based on play-style. Success in these more complex domains would allow for the integration of our model into current and future games where the complete state is not available. Additionally, transformer-based models have demonstrated remarkable success in natural language processing tasks [Wolf *et al.* 2020] and could be adapted to replace our LSTM autoencoder, potentially improving the model’s ability to cluster trajectories based on play-style. There has also been some work on applying transformers to images, showing further potential for adapting our model to image-based datasets [Wu *et al.* 2021]. Similarly, there have been encouraging efforts to explore contrastive losses in the latent space as a means of promoting clustering, and this avenue of research warrants further investigation [Guo *et al.* 2022]. Lastly, rather than relying exclusively on gameplay data, future research could explore the use of multi-modal data (e.g., audio, biometric, and contextual data) to better identify an individual’s play-style.

9.2.2 Game Modeller (GM)

Inverse reinforcement learning (IRL) [Ng *et al.* 2000] is a promising approach that could potentially improve or replace the behavioural cloning aspect of our GM. IRL aims to learn the underlying reward function of an expert from its demonstrated behaviour, and then use this reward function to generate a new behaviour that is similar to the expert’s behaviour. By doing so, IRL can produce behaviours that go beyond simple mimicry, as it learns the reasoning behind the expert’s behaviour. A potential advantage of IRL over behavioural cloning is that IRL can handle situations where the expert’s behaviour is sub-optimal or imperfect. This is because IRL learns the underlying reward function, whereas behavioural cloning only learns from the expert’s behaviour [Codevilla *et al.* 2019]. However, IRL is also computationally more expensive than behavioural cloning and requires a larger amount of data not usually available in video game domains.

9.2.3 Player Modeller (PM)

Incorporating more information about the identified play-style characteristics could potentially improve the online prediction accuracy of the model. Currently, the play-style identifier is used as a feature in the overall model, but additional information about the characteristics of each play-style could provide a more detailed understanding of a player’s behaviour. For example, if a play-style is identified as aggressive, the model could benefit from additional information about the player’s tendency to take risks, their response to pressure, or their preferred offensive strategies. By incorporating more detailed information about play-style characteristics, the model could potentially make more accurate predictions about a player’s behaviour.

9.2.4 Advisor (AD)

Future research in the area of advising can explore various avenues to enhance the effectiveness of advising. Particularly, testing this system with human participants has the potential to provide valuable insights into its effectiveness and practical application in real-world settings. However, it would require careful planning and execution to ensure ethical considerations are taken into account and to optimise the system for use with human-generated data.

Another key challenge in automated advice generation is knowing when to provide advice and when to let players learn on their own. Future research could explore how to incorporate machine learning algorithms to identify the most opportune moments to provide advice and ensure that the advice is helpful and not overwhelming [Judah *et al.* 2010]. It is important to consider the timing and frequency of advice delivery, as well as the type of advice given. For example, should advice be given proactively or reactively? Should advice be given in real-time or after a game session? One potential solution could be to use a reinforcement learning agent that learns when to give advice based on user behaviour. The actions of the agent would be to give or not give advice, and the reward function would depend on whether or not the user follows it. This could be a promising approach to tailor advice to individual users and improve the overall effectiveness of tutorial systems.

Moreover, research can investigate how to tailor advice to the player's experience level, as novice players may require different types of advice compared to expert players. Another area of research can focus on the design of the advice itself. The language, tone, and specificity of the advice can impact its effectiveness [Torrey *et al.* 2013]. Research can investigate how to design advice that is clear, concise, and actionable. Another avenue of research can be to explore how to motivate players to follow the advice given. Research can investigate how to frame the advice in a way that is motivating and encourages players to engage with it. Recent advancements in large language models could be leveraged to generate natural language advice for users, rather than relying on hard-coded or grammar-based techniques. Large language models such as GPT-3 [Floridi and Chiriatti 2020] have shown impressive capabilities in generating human-like text, which could potentially be applied to the generation of personalised advice for users. By training the language model on a large dataset of user behaviour and expert knowledge, the model could learn to generate advice that is more personalised, relevant, and natural-sounding to the user. This could potentially lead to more effective tutorial systems and greater user engagement.

Furthermore, research can explore how to integrate game design elements that incentivise players to follow the advice given, such as through rewards or other game-based mechanics [Blair 2011]. The advice generator can be integrated into the game in various ways. For instance, it can be implemented as a pop-up window that appears when a player spends too much time on a particular puzzle or task. Alternatively, it can be integrated into the game's reward system, where players can earn hints and tips as they progress through the levels. Additionally, in a broader sense research can investigate how advising can be integrated into other learning experiences. For example, research can explore how advising can be integrated with other forms of feedback, such as progress tracking and assessments [Robertson and Howells 2008]. Since our research explored how to personalise advice to individual players based on their play-style, we could incorporate other factors of an individual such as their age, experience from other games or desire to learn. In particular Tekofsky *et al.* [2015] demonstrated that age has a significant influence on play-style. Thus, including this particular feature information could potentially enhance both the identification process of play-style and the generated advice. Finally, future research could explore the use of collaborative advice generation, where players are actively involved in generating and refining the advice they receive.

All these avenues have the potential to enhance the overall user experience and lead to increased engagement and motivation to complete the game. Conducting further research and implementing the

aforementioned improvements would represent a significant step forward in the development of an effective automated system for tailored advice generation. Such a system has the potential to provide valuable guidance and support to users, facilitating their improvement in various domains, not just video games.

9.3 Discussion

Automated tailored advice generation for video games has the potential to greatly impact the way people learn to play video games. By learning in their own style, players can feel more confident in their abilities, leading to a more positive experience overall. The current state of tutorial systems in games is often quite simplistic and does not provide a personalised or adaptive experience. Many games still rely on text-based tutorials or video tutorials that are often quickly forgotten or become irrelevant as players progress. This can be frustrating for players who are trying to learn a new game, and it can also be a barrier to entry for new players who may become overwhelmed or frustrated by the complexity of a game.

However, with the development of automated systems for tailored advice generation, there is the potential to drastically change tutorial systems and provide a personalised, evolving teacher for players. By identifying a player's unique play-style characteristics, a system can provide tailored advice that is specific to the individual, rather than generic advice that may not apply to every player. Moreover, as the system continually learns from the player's behaviour and progress, it can adapt and provide new advice as needed, ensuring that the player is always receiving the most relevant guidance. By providing a personalised and evolving teacher, players may be more likely to stick with a game and continue to improve, even if they struggle at first. This approach may be particularly effective for novice players who may be intimidated by complex games or overwhelmed by the amount of information they need to learn. By providing tailored advice that is specific to their play-style and learning style, they may feel more confident and motivated to continue playing and improving. Overall, the development of automated systems for tailored advice generation has the potential to greatly enhance the tutorial experience in games.

Additionally, this approach to learning has the potential to transfer to other domains where learning play-style characteristics can be applied to improve how people learn with respect to their own preferred style. One potential application for this technology is in educational settings. By identifying a student's preferred learning style, educators can better tailor their teaching methods to fit the student's needs. This could lead to more efficient learning and higher retention of information. Additionally, this technology could be applied in sports and other physical activities, where understanding and improving upon one's own style is crucial for success.

In conclusion, automated tailored advice generation has the potential to improve the way people learn and improve their skills in a variety of domains. By identifying and utilising a person's preferred learning or play-style, this technology can improve the efficiency and effectiveness of learning. Additionally, learning in one's own style can lead to a more positive and engaging learning experience, which is likely to lead to more sustained improvements.

References

- [Abbeel and Ng 2004] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.
- [Albrecht and Stone 2018] Stefano V Albrecht and Peter Stone. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence*, 258:66–95, 2018.
- [Aleahmad *et al.* 2008] Turadg Aleahmad, Vincent Alevan, and Robert Kraut. Open community authoring of targeted worked example problems. In *Intelligent Tutoring Systems: 9th International Conference, ITS 2008, Montreal, Canada, June 23-27, 2008 Proceedings 9*, pages 216–227. Springer, 2008.
- [Allahyari *et al.* 2017] Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assefi, Saied Safaei, Elizabeth D Trippe, Juan B Gutierrez, and Krys Kochut. A brief survey of text mining: Classification, clustering and extraction techniques. *arXiv preprint arXiv:1707.02919*, 2017.
- [Amari 1993] Shun-ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196, 1993.
- [Andersen *et al.* 2012] Erik Andersen, Eleanor O’rourke, Yun-En Liu, Rich Snider, Jeff Lowdermilk, David Truong, Seth Cooper, and Zoran Popovic. The impact of tutorials on games of varying complexity. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 59–68, 2012.
- [Anderson *et al.* 1985] John R Anderson, C Franklin Boyle, and Brian J Reiser. Intelligent tutoring systems. *Science*, 228(4698):456–462, 1985.
- [Anderson 2013] John R Anderson. *The architecture of cognition*. Psychology Press, 2013.
- [Andrade *et al.* 2005] Gustavo Andrade, Geber Ramalho, Hugo Santana, and Vincent Corruble. Extending reinforcement learning to provide dynamic game balancing. In *Proceedings of the Workshop on Reasoning, Representation, and Learning in Computer Games, 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 7–12, 2005.
- [Andreas *et al.* 2016] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 39–48, 2016.
- [Arendse *et al.* 2022] Lindsay Arendse, Branden Ingram, and Benjamin Rosman. Real time in-game playstyle classification using a hybrid probabilistic supervised learning approach. In *The Third Southern African Conference for AI Research Proceedings. Part of the book series: Communications in Computer and Information Science, Springer, December 2022*. Springer, 2022.

- [Argall *et al.* 2009] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.
- [Arrabales *et al.* 2012] Raúl Arrabales, Jorge Muñoz, Agapito Ledezma, German Gutierrez, and Araceli Sanchis. A machine consciousness approach to the design of human-like bots. *Believable Bots: Can Computers Play Like People?*, pages 171–191, 2012.
- [Arras *et al.* 2017] Leila Arras, Franziska Horn, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. ” what is relevant in a text document?”: An interpretable machine learning approach. *PloS one*, 12(8):e0181142, 2017.
- [Arzate Cruz and Ramirez Uresti 2018] Christian Arzate Cruz and Jorge Adolfo Ramirez Uresti. Hrlb: A reinforcement learning based framework for believable bots. *Applied Sciences*, 8(12):2453, 2018.
- [Baarslag *et al.* 2016] Tim Baarslag, Mark JC Hendriks, Koen V Hindriks, and Catholijn M Jonker. Learning about the opponent in automated bilateral negotiation: a comprehensive survey of opponent modeling techniques. *Autonomous Agents and Multi-Agent Systems*, 30:849–898, 2016.
- [Bahad *et al.* 2019] Pritika Bahad, Preeti Saxena, and Raj Kamal. Fake news detection using bi-directional lstm-recurrent neural network. *Procedia Computer Science*, 165:74–82, 2019.
- [Bakker *et al.* 1996] Paul Bakker, Yasuo Kuniyoshi, et al. Robot see, robot do: An overview of robot imitation. In *AISB96 Workshop on Learning in Robots and Animals*, volume 5. Citeseer, 1996.
- [Bakkes *et al.* 2012] Sander CJ Bakkes, Pieter HM Spronck, and Giel van Lankveld. Player behavioural modelling for video games. *Entertainment Computing*, 3(3):71–79, 2012.
- [Bartle 1996] Richard Bartle. Hearts, clubs, diamonds, spades: Players who suit muds. *Journal of MUD research*, 1(1):19, 1996.
- [Bartle 2004] Richard A Bartle. *Designing virtual worlds*. New Riders, 2004.
- [Bellman 1957] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.
- [Berndt and Clifford 1994] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, USA:, 1994.
- [Berner *et al.* 2019] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.
- [Beukman *et al.* 2022] Michael Beukman, Christopher W Cleghorn, and Steven James. Procedural content generation using neuroevolution and novelty search for diverse video game levels. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1028–1037, 2022.
- [Birk *et al.* 2017] Max V Birk, Maximilian A Friehs, and Regan L Mandryk. Age-based preferences and player experience: a crowdsourced cross-sectional study. In *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, pages 157–170, 2017.
- [Bishop and Nasrabadi 2006] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.
- [Blair 2011] Lucas Blair. The use of video game achievements to enhance player performance, self-efficacy, and motivation. 2011.

- [Botvinick *et al.* 2019] Matthew Botvinick, Sam Ritter, Jane X Wang, Zeb Kurth-Nelson, Charles Blundell, and Demis Hassabis. Reinforcement learning, fast and slow. *Trends in cognitive sciences*, 23(5):408–422, 2019.
- [Bourlard and Kamp 1988] Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4-5):291–294, 1988.
- [Bouza *et al.* 2008] Amancio Bouza, Gerald Reif, Abraham Bernstein, and Harald Gall. Semtree: ontology-based decision tree algorithm for recommender systems. 2008.
- [Bowman *et al.* 2015] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.
- [Brisson *et al.* 2012] António Brisson, Gonçalo Pereira, Rui Prada, Ana Paiva, Sandy Louchart, Neil Suttie, Theo Lim, Ricardo Lopes, Rafael Bidarra, Francesco Bellotti, et al. Artificial intelligence and personalization opportunities for serious games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 8, pages 51–57, 2012.
- [Bullough Jr *et al.* 2003] Robert V Bullough Jr, Janet Young, James R Birrell, D Cecil Clark, M Winston Egan, Lynnette Erickson, Marti Frankovich, Joanne Brunetti, and Myra Welling. Teaching with a peer: A comparison of two models of student teaching. *Teaching and teacher education*, 19(1):57–73, 2003.
- [Cannel and Markovitch 1993] David Cannel and Shaul Markovitch. Learning models of opponent’s strategy game playing. In *Proceedings of the 1993 AAAI Fall Symposium on Games: Learning and Planning*, pages 140–147, 1993.
- [Carpenter and Grossberg 1987] Gail A Carpenter and Stephen Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer vision, graphics, and image processing*, 37(1):54–115, 1987.
- [Chai and Draxler 2014] Tianfeng Chai and Roland R Draxler. Root mean square error (rmse) or mean absolute error (mae)?—arguments against avoiding rmse in the literature. *Geoscientific model development*, 7(3):1247–1250, 2014.
- [Char *et al.* 2018] Danton S Char, Nigam H Shah, and David Magnus. Implementing machine learning in health care—addressing ethical challenges. *The New England journal of medicine*, 378(11):981, 2018.
- [Charles *et al.* 2005] Darryl Charles, Michael Mcneill, Moira McAlister, Michaela Black, Adrian Moore, Karl Stringer, Julian Kücklich, and Aphra Kerr. Player-centred game design: Player modelling and adaptive digital games. 2005.
- [Cheung Chiu and Webb 1998] Bark Cheung Chiu and Geoffrey I Webb. Using decision trees for agent modeling: improving prediction performance. *User Modeling and User-Adapted Interaction*, 8:131–152, 1998.
- [Chowdary *et al.* 2014] N Sunil Chowdary, DS Prasanna, and P Sudhakar. Evaluating and analyzing clusters in data mining using different algorithms. 2014.
- [Chung *et al.* 2014] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.
- [Clouse 1997] Jeffery A Clouse. On integrating apprentice learning and reinforcement learning. 1997.

- [Cobbe *et al.* 2019] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 1282–1289. PMLR, 2019.
- [Codevilla *et al.* 2019] Felipe Codevilla, Eder Santana, Antonio M López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9329–9338, 2019.
- [Coggan 2004] Melanie Coggan. Exploration and exploitation in reinforcement learning. *Research supervised by Prof. Doina Precup, CRA-W DMP Project at McGill University*, 2004.
- [Coleman and Andrews 1979] Guy Barrett Coleman and Harry C Andrews. Image segmentation by clustering. *Proceedings of the IEEE*, 67(5):773–785, 1979.
- [Conroy *et al.* 2012] David Conroy, Peta Wyeth, and Daniel Johnson. Spotting the difference: Identifying player opponent preferences in fps games. In *Entertainment Computing-ICEC 2012: 11th International Conference, ICEC 2012, Bremen, Germany, September 26-29, 2012. Proceedings 11*, pages 114–121. Springer, 2012.
- [De Marzo and Servedio 2022] Giordano De Marzo and Vito DP Servedio. Quantifying the complexity and similarity of chess openings using online chess community data. *arXiv preprint arXiv:2206.14312*, 2022.
- [Dey and Salem 2017] Rahul Dey and Fathi M Salem. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWS-CAS)*, pages 1597–1600. IEEE, 2017.
- [Drachen *et al.* 2009] Anders Drachen, Alessandro Canossa, and Georgios N Yannakakis. Player modeling using self-organization in tomb raider: Underworld. In *2009 IEEE symposium on computational intelligence and games*, pages 1–8. IEEE, 2009.
- [Drachen *et al.* 2014] Anders Drachen, Matthew Yancey, John Maguire, Derrek Chu, Iris Yuhui Wang, Tobias Mahlmann, Matthias Schubert, and Diego Klabajan. Skill-based differences in spatio-temporal team behaviour in defence of the ancients 2 (dota 2). In *2014 IEEE Games Media Entertainment*, pages 1–8. IEEE, 2014.
- [Dunn 1984] Rita Dunn. Learning style: State of the science. *Theory into practice*, 23(1):10–19, 1984.
- [Duvalllet *et al.* 2013] Felix Duvalllet, Thomas Kollar, and Anthony Stentz. Imitation learning for natural language direction following through unknown environments. In *2013 IEEE International Conference on Robotics and Automation*, pages 1047–1053. IEEE, 2013.
- [Edwards and Cavalli-Sforza 1965] Anthony WF Edwards and Luigi Luca Cavalli-Sforza. A method for cluster analysis. *Biometrics*, pages 362–375, 1965.
- [Eggert *et al.* 2015] Christoph Eggert, Marc Herrlich, Jan Smeddinck, and Rainer Malaka. Classification of player roles in the team-based multi-player game dota 2. In *International Conference on Entertainment Computing*, pages 112–125. Springer, 2015.
- [Ehsan *et al.* 2018a] Upol Ehsan, Brent Harrison, Larry Chan, and Mark O Riedl. Rationalization: A neural machine translation approach to generating natural language explanations. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pages 81–87. ACM, 2018.
- [Ehsan *et al.* 2018b] Upol Ehsan, Pradyumna Tambwekar, Larry Chan, Brent Harrison, and Mark O Riedl. Learning to generate natural language rationales for game playing agents. In *Joint Proceedings of the AIIDE 2018 Workshops*, volume 2282, page 1, 2018.

- [Feiman-Nemser 1983] Sharon Feiman-Nemser. *Learning to teach.*, 1983.
- [Fisher 1987] Douglas H Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine learning*, 2(2):139–172, 1987.
- [Floridi and Chiriatti 2020] Luciano Floridi and Massimo Chiriatti. Gpt-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30:681–694, 2020.
- [Fong and Vedaldi 2017] Ruth C Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3429–3437, 2017.
- [Fortuin *et al.* 2018] Vincent Fortuin, Matthias Hüser, Francesco Locatello, Heiko Strathmann, and Gunnar Rätsch. Som-vae: Interpretable discrete representation learning on time series. *arXiv preprint arXiv:1806.02199*, 2018.
- [Fraley and Raftery 1998] Chris Fraley and Adrian E Raftery. How many clusters? which clustering method? answers via model-based cluster analysis. *The computer journal*, 41(8):578–588, 1998.
- [Frogner *et al.* 2015] Charlie Frogner, Chiyuan Zhang, Hossein Mobahi, Mauricio Araya-Polo, and Tomaso Poggio. Learning with a wasserstein loss. *arXiv preprint arXiv:1506.05439*, 2015.
- [Garvin and Margolis 2015] David A Garvin and Joshua D Margolis. The art of giving and receiving advice. *Harvard business review*, 93(1):14, 2015.
- [Gershenfeld and Weigend 1993] Neil A Gershenfeld and Andreas S Weigend. *The future of time series*. Xerox Corporation, Palo Alto Research Center Palo Alto, CA, USA, 1993.
- [Ghader and Monz 2017] Hamidreza Ghader and Christof Monz. What does attention in neural machine translation pay attention to? *arXiv preprint arXiv:1710.03348*, 2017.
- [Gong *et al.* 2019] Dong Gong, Lingqiao Liu, Vuong Le, Budhaditya Saha, Moussa Reda Mansour, Svetha Venkatesh, and Anton van den Hengel. Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1705–1714, 2019.
- [Gorman and Humphrys 2007] Bernard Gorman and Mark Humphrys. Imitative learning of combat behaviours in first-person computer games. *Proceedings of CGAMES*, 2007.
- [Gow *et al.* 2012] Jeremy Gow, Robin Baumgarten, Paul Cairns, Simon Colton, and Paul Miller. Unsupervised modeling of player style with lda. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(3):152–166, 2012.
- [Grabusts 2011] Peter Grabusts. The choice of metrics for clustering algorithms. In *ENVIRONMENT. TECHNOLOGIES. RESOURCES. Proceedings of the International Scientific and Practical Conference*, volume 2, pages 70–76, 2011.
- [Graesser *et al.* 2012] Arthur C Graesser, Mark W Conley, and Andrew Olney. Intelligent tutoring systems. *APA educational psychology handbook, Vol 3: Application to learning and teaching.*, pages 451–473, 2012.
- [Grapevine 2020] Grapevine. *The Famous (Infamous?) Archetype Triangle*. <https://smashlabs.design.blog/2020/05/03/archetype-triangle/>, 2020. Online; accessed 2022-07-21.

- [Graves *et al.* 2013] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, 2013.
- [Graving and Couzin 2020] Jacob M Graving and Iain D Couzin. Vae-sne: a deep generative model for simultaneous dimensionality reduction and clustering. *BioRxiv*, pages 2020–07, 2020.
- [Greydanus *et al.* 2017] Sam Greydanus, Anurag Koul, Jonathan Dodge, and Alan Fern. Visualizing and understanding atari agents. *arXiv preprint arXiv:1711.00138*, 2017.
- [Gunning 2017] David Gunning. Explainable artificial intelligence (xai). *Defense Advanced Research Projects Agency (DARPA), nd Web*, 2, 2017.
- [Guo *et al.* 2017] Xifeng Guo, Xinwang Liu, En Zhu, and Jianping Yin. Deep clustering with convolutional autoencoders. In *Neural Information Processing: 24th International Conference, ICONIP 2017, Guangzhou, China, November 14-18, 2017, Proceedings, Part II 24*, pages 373–382. Springer, 2017.
- [Guo *et al.* 2022] Yuanfan Guo, Minghao Xu, Jiawen Li, Bingbing Ni, Xuanyu Zhu, Zhenbang Sun, and Yi Xu. Hcsc: hierarchical contrastive selective coding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9706–9715, 2022.
- [Guzdial and Riedl 2016] Matthew Guzdial and Mark Riedl. Game level generation from gameplay videos. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.
- [Harmer *et al.* 2018] Jack Harmer, Linus Gisslén, Jorge del Val, Henrik Holst, Joakim Bergdahl, Tom Olsson, Kristoffer Sjöö, and Magnus Nordin. Imitation learning with concurrent actions in 3d games. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2018.
- [Hartigan and Wong 1979] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.
- [Harvey *et al.* 2000] Nigel Harvey, Clare Harries, and Ilan Fischer. Using advice and assessing its quality. *Organizational behavior and human decision processes*, 81(2):252–273, 2000.
- [Hastings *et al.* 2009] Erin J Hastings, Ratan K Guha, and Kenneth O Stanley. Evolving content in the galactic arms race video game. In *2009 IEEE Symposium on Computational Intelligence and Games*, pages 241–248. IEEE, 2009.
- [Haykin 1994] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.
- [He *et al.* 2010] Xiaofei He, Deng Cai, Yuanlong Shao, Hujun Bao, and Jiawei Han. Laplacian regularized gaussian mixture model for data clustering. *IEEE Transactions on Knowledge and Data Engineering*, 23(9):1406–1418, 2010.
- [He *et al.* 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [Hendrycks *et al.* 2019] Dan Hendrycks, Kimin Lee, and Mantas Mazeika. Using pre-training can improve model robustness and uncertainty. In *International Conference on Machine Learning*, pages 2712–2721. PMLR, 2019.

- [Herrmann 2022] Daniel A Herrmann. Prediction with expert advice applied to the problem of prediction with expert advice. *Synthese*, 200:315, 2022.
- [Hester *et al.* 2018] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.
- [Ho and Ermon 2016] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016.
- [Hochreiter and Schmidhuber 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [Hochreiter 1998] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.
- [Holmgard *et al.* 2014] Christoffer Holmgard, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis. Evolving personas for player decision modeling. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8, 2014.
- [Hu *et al.* 2017] Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to reason: End-to-end module networks for visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 804–813, 2017.
- [Hu *et al.* 2019] Zhiting Hu, Bowen Tan, Russ R Salakhutdinov, Tom M Mitchell, and Eric P Xing. Learning data manipulation for augmentation and weighting. *Advances in Neural Information Processing Systems*, 32, 2019.
- [Hua *et al.* 2019] Yuxiu Hua, Zhifeng Zhao, Rongpeng Li, Xianfu Chen, Zhiming Liu, and Honggang Zhang. Deep learning with long short-term memory for time series prediction. *IEEE Communications Magazine*, 57(6):114–119, 2019.
- [Huang *et al.* 2019] Xuan Huang, Lei Wu, and Yinsong Ye. A review on dimensionality reduction techniques. *International Journal of Pattern Recognition and Artificial Intelligence*, 33(10):1950017, 2019.
- [Hughes and Bartlett 2002] Mike D Hughes and Roger M Bartlett. The use of performance indicators in performance analysis. *Journal of sports sciences*, 20(10):739–754, 2002.
- [Hussein *et al.* 2017] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.
- [Ingram *et al.* 2022a] Branden Ingram, Benjamin Rosman, Richard Klein, and Clint van Alten. Play-style identification through deep unsupervised clustering of trajectories. In *2022 IEEE Conference on Games (CoG)*. IEEE, 2022.
- [Ingram *et al.* 2022b] Branden Ingram, Benjamin Rosman, Clint van Alten, and Richard Klein. Improved action prediction through multiple model processing of player trajectories. In *2022 IEEE Conference on Games (CoG)*, pages 548–551. IEEE, 2022.
- [Ingram *et al.* 2023a] Branden Ingram, Benjamin Rosman, Clint van Alten, and Richard Klein. Creating diverse play-style-centric agents through behavioural cloning. In *Proceedings of the Eighteenth*

AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2023, Salt Lake City, UT, USA. AAAI Press, 2023.

- [Ingram *et al.* 2023b] Branden Ingram, Clint van Alten, Richard Klein, and Benjamin Rosman. Generating interpretable play-style descriptions through deep unsupervised clustering of trajectories. *IEEE Transactions on Games*, 2023.
- [Ingram 2021] Branden Ingram. Generating tailored advice in video games through play-style identification and player modelling. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 17, pages 228–231, 2021.
- [Isbister and Schaffer 2008] Katherine Isbister and Noah Schaffer. *Game usability: Advancing the player experience*. CRC press, 2008.
- [Jacob *et al.* 2022] Athul Paul Jacob, David J Wu, Gabriele Farina, Adam Lerer, Hengyuan Hu, Anton Bakhtin, Jacob Andreas, and Noam Brown. Modeling strong and human-like gameplay with kl-regularized search. In *International Conference on Machine Learning*, pages 9695–9728. PMLR, 2022.
- [Jaderberg *et al.* 2019] Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.
- [Johnson *et al.* 2017] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2901–2910, 2017.
- [Judah *et al.* 2010] Kshitij Judah, Saikat Roy, Alan Fern, and Thomas Dietterich. Reinforcement learning via practice and critique advice. In *Proceedings of the AAAI conference on artificial intelligence*, volume 24, pages 481–486, 2010.
- [Kaelbling *et al.* 1996] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.
- [Kamnitsas *et al.* 2018] Konstantinos Kamnitsas, Daniel Castro, Loic Le Folgoc, Ian Walker, Ryutaro Tanno, Daniel Rueckert, Ben Glocker, Antonio Criminisi, and Aditya Nori. Semi-supervised learning via compact latent space clustering. In *International conference on machine learning*, pages 2459–2468. PMLR, 2018.
- [Kartsanis and Murzyn 2016] Nikolaos Kartsanis and Eva Murzyn. Me, my game-self, and others: A qualitative exploration of the game-self. In *2016 International Conference on Interactive Technologies and Games (ITAG)*, pages 29–35. IEEE, 2016.
- [Keogh and Lin 2005] Eamonn Keogh and Jessica Lin. Clustering of time-series subsequences is meaningless: implications for previous and future research. *Knowledge and information systems*, 8(2):154–177, 2005.
- [Khalifa *et al.* 2016] Ahmed Khalifa, Aaron Isaksen, Julian Togelius, and Andy Nealen. Modifying mcts for human-like general video game playing. 2016.
- [Khoshkangini *et al.* 2018] Reza Khoshkangini, Santiago Ontanón, Annapaola Marconi, and Jichen Zhu. Dynamically extracting play style in educational games. *EUROSIS proceedings, GameOn*, 2018.

- [Kim *et al.* 2013] Beomjoon Kim, Amir-massoud Farahmand, Joelle Pineau, and Doina Precup. Approximate policy iteration with demonstration data. *RLDM 2013*, page 168, 2013.
- [Kindermans *et al.* 2017] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. The (un) reliability of saliency methods. *arXiv preprint arXiv:1711.00867*, 2017.
- [Kingma and Welling 2013] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- [Kirman and Lawson 2009] Ben Kirman and Shaun Lawson. Hardcore classification: Identifying play styles in social games using network analysis. In *Entertainment Computing–ICEC 2009: 8th International Conference, Paris, France, September 3-5, 2009. Proceedings* 8, pages 246–251. Springer, 2009.
- [Kisilevich *et al.* 2009] Slava Kisilevich, Florian Mansmann, Mirco Nanni, and Salvatore Rinzivillo. Spatio-temporal clustering. In *Data mining and knowledge discovery handbook*, pages 855–874. Springer, 2009.
- [Kober *et al.* 2013] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.
- [Kohonen 1990] Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.
- [Koster 2013] Raph Koster. *Theory of fun for game design*. ” O’Reilly Media, Inc.”, 2013.
- [Kriegel *et al.* 2011] Hans-Peter Kriegel, Peer Kröger, Jörg Sander, and Arthur Zimek. Density-based clustering. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, 1(3):231–240, 2011.
- [Krizhevsky *et al.* 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [Kuhn 1955] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.
- [Lagoudakis and Parr 2003] Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149, 2003.
- [Lange *et al.* 2021] Andrey Lange, Andrey Somov, Anton Stepanov, and Evgeny Burnaev. Building a behavioral profile and assessing the skill of video game players. *IEEE Sensors Journal*, 22(1):481–488, 2021.
- [Liao 2005] T Warren Liao. Clustering of time series data—a survey. *Pattern recognition*, 38(11):1857–1874, 2005.
- [Lillicrap *et al.* 2015] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.
- [Lin 1992] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.
- [Lin 1993] Long-Ji Lin. *Reinforcement learning for robots using neural networks*. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

- [Lindberg and Laine 2016] Renny SN Lindberg and Teemu Henrikki Laine. Detecting play and learning styles for adaptive educational games. In *CSEDU (1)*, pages 181–189, 2016.
- [Lindley 2005] Craig A Lindley. Story and narrative structures in computer games. *Bushoff, Brunhild. ed*, 2005.
- [Liu *et al.* 2013] Siming Liu, Christopher Ballinger, and Sushil J Louis. Player identification from rts game replays. *Proceedings of the 28th CATA*, pages 313–317, 2013.
- [Liu *et al.* 2019] Yang Liu, Eunice Jun, Qisheng Li, and Jeffrey Heer. Latent space cartography: Visual analysis of vector space embeddings. In *Computer graphics forum*, volume 38, pages 67–78. Wiley Online Library, 2019.
- [Lopes *et al.* 2007] Manuel Lopes, Francisco S Melo, and Luis Montesano. Affordance-based imitation learning in robots. In *2007 IEEE/RSJ international conference on intelligent robots and systems*, pages 1015–1021. IEEE, 2007.
- [Luckin and others 2007] R Luckin et al. Modeling learning patterns of students with a tutoring system using hidden markov models. *Artificial intelligence in education: Building technology rich learning contexts that work*, 158:238, 2007.
- [Madiraju *et al.* 2018] Naveen Sai Madiraju, Seid M Sadat, Dmitry Fisher, and Homa Karimabadi. Deep temporal clustering: Fully unsupervised learning of time-domain features. *arXiv preprint arXiv:1802.01059*, 2018.
- [Malinowski *et al.* 2015] Mateusz Malinowski, Marcus Rohrbach, and Mario Fritz. Ask your neurons: A neural-based approach to answering questions about images. In *Proceedings of the IEEE international conference on computer vision*, pages 1–9, 2015.
- [Malone 1980] Thomas W Malone. What makes things fun to learn? heuristics for designing instructional computer games. In *Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems*, pages 162–169. ACM, 1980.
- [Marom and Rosman 2018] Ofir Marom and Benjamin Rosman. Belief reward shaping in reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.
- [Martens and Dardenne 1998] Harald A Martens and Pierre Dardenne. Validation and verification of regression in small data sets. *Chemometrics and intelligent laboratory systems*, 44(1-2):99–121, 1998.
- [Martin 2012] Crystle Martin. Video games, identity, and the constellation of information. *Bulletin of Science, Technology & Society*, 32(5):384–392, 2012.
- [Masood and Khan 2015] Muhammad Ali Masood and MNA Khan. Clustering techniques in bioinformatics. *IJ Modern Education and Computer Science*, 1:38–46, 2015.
- [McClelland *et al.* 1986] James L McClelland, David E Rumelhart, PDP Research Group, et al. *Parallel distributed processing*, volume 2. MIT press Cambridge, MA, 1986.
- [McIlroy-Young *et al.* 2020] Reid McIlroy-Young, Siddhartha Sen, Jon Kleinberg, and Ashton Anderson. Aligning superhuman ai with human behavior: Chess as a model system. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1677–1687, 2020.

- [Melo *et al.* 2008] Francisco S Melo, Sean P Meyn, and M Isabel Ribeiro. An analysis of reinforcement learning with function approximation. In *Proceedings of the 25th international conference on Machine learning*, pages 664–671, 2008.
- [Meng *et al.* 2017] Qinxue Meng, Daniel Catchpoole, David Skillicom, and Paul J Kennedy. Relational autoencoder for feature extraction. In *2017 International joint conference on neural networks (IJCNN)*, pages 364–371. IEEE, 2017.
- [Mensink *et al.* 2021] Thomas Mensink, Jasper Uijlings, Alina Kuznetsova, Michael Gygli, and Vittorio Ferrari. Factors of influence for transfer learning across diverse appearance domains and task types. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(12):9298–9314, 2021.
- [Milligan and Cooper 1987] Glenn W Milligan and Martha C Cooper. Methodology review: Clustering methods. *Applied psychological measurement*, 11(4):329–354, 1987.
- [Mitchell 2009] William A Mitchell. Multi-behavioral strategies in a predator–prey game: an evolutionary algorithm analysis. *Oikos*, 118(7):1073–1083, 2009.
- [Mnih *et al.* 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- [Mnih *et al.* 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.
- [Moerland *et al.* 2023] Thomas M Moerland, Joost Broekens, Aske Plaat, Catholijn M Jonker, et al. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, 2023.
- [Muñoz *et al.* 2013] Jorge Muñoz, German Gutierrez, and Araceli Sanchis. Towards imitation of human driving style in car racing games. In *Believable bots*, pages 289–313. Springer, 2013.
- [Murtagh and Contreras 2012] Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.
- [Myung 2003] In Jae Myung. Tutorial on maximum likelihood estimation. *Journal of mathematical Psychology*, 47(1):90–100, 2003.
- [Nanni 2002] Mirco Nanni. *Clustering Methods for Spatio-temporal Data: Ph. D. Thesis*. 2002.
- [Nashed and Zilberstein 2022] Samer Nashed and Shlomo Zilberstein. A survey of opponent modeling in adversarial domains. *Journal of Artificial Intelligence Research*, 73:277–327, 2022.
- [Nebel *et al.* 2016] Steve Nebel, Sascha Schneider, and Günter Daniel Rey. Mining learning and crafting scientific experiments: a literature review on the use of minecraft in education and research. *Journal of Educational Technology & Society*, 19(2):355–366, 2016.
- [Needleman and Wunsch 1970] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.
- [Ng *et al.* 2000] Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

- [Oh *et al.* 2015] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. *Advances in neural information processing systems*, 28, 2015.
- [Olesen *et al.* 2008] Jacob Kaae Olesen, Georgios N Yannakakis, and John Hallam. Real-time challenge balance in an rts game using rtneat. In *2008 IEEE Symposium On Computational Intelligence and Games*, pages 87–94. IEEE, 2008.
- [OpenAI 2007] OpenAI. *About OpenAI*. <https://openai.com/>, 2007. Online; accessed 2019-07-21.
- [Papamichail and French 2003] K Nadia Papamichail and Simon French. Explaining and justifying the advice of a decision support system: a natural language generation approach. *Expert Systems with Applications*, 24(1):35–48, 2003.
- [Pearce and Zhu 2022] Tim Pearce and Jun Zhu. Counter-strike deathmatch with large-scale behavioural cloning. In *2022 IEEE Conference on Games (CoG)*, pages 104–111. IEEE, 2022.
- [Perez-Liebana *et al.* 2021] Diego Perez-Liebana, Cristina Guerrero-Romero, Alexander Dockhorn, Linjie Xu, Jorge Hurtado, and Dominik Jeurissen. Generating diverse and competitive play-styles for strategy games. In *2021 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2021.
- [Phillips *et al.* 2010] Elissa Phillips, Keith Davids, Ian Renshaw, and Marc Portus. Expert performance in sport and the dynamics of talent development. *Sports medicine*, 40:271–283, 2010.
- [Porter *et al.* 2010] Guy Porter, Vladan Starcevic, David Berle, and Pauline Fenech. Recognizing problem video game use. *Australian & New Zealand Journal of Psychiatry*, 44(2):120–128, 2010.
- [Przybylski *et al.* 2010] Andrew K Przybylski, C Scott Rigby, and Richard M Ryan. A motivational model of video game engagement. *Review of general psychology*, 14(2):154–166, 2010.
- [Ramachandran and Amir 2007] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. In *IJCAI*, volume 7, pages 2586–2591, 2007.
- [Ramanishka *et al.* 2017] Vasili Ramanishka, Abir Das, Jianming Zhang, and Kate Saenko. Top-down visual saliency guided by captions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7206–7215, 2017.
- [Ranchod *et al.* 2015] Pravesh Ranchod, Benjamin Rosman, and George Konidaris. Nonparametric bayesian reward segmentation for skill discovery using inverse reinforcement learning. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 471–477. IEEE, 2015.
- [Ribeiro *et al.* 2016] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. ACM, 2016.
- [Riezebos *et al.* 2021] Mark Riezebos, Michel van de Velden, S Ilker Birbil, Supervisors ORTEC Sports BV, Ruud van der Knaap, and Bertus Talsma. *Identifying Play Styles of Football Players Based on Match Event Data*. PhD thesis, Erasmus School of Economics, Erasmus University Rotterdam Rotterdam, The ..., 2021.
- [Ritter *et al.* 2003] Steven Ritter, Stephen B Blessing, and Leslie Wheeler. Authoring tools for component-based learning environments. *Authoring Tools for Advanced Technology Learning Environments: Toward Cost-Effective Adaptive, Interactive and Intelligent Educational Software*, pages 467–489, 2003.

- [Robertson and Howells 2008] Judy Robertson and Cathrin Howells. Computer game design: Opportunities for successful learning. *Computers & Education*, 50(2):559–578, 2008.
- [Ross *et al.* 2011] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.
- [Ruder 2016] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.
- [Rumelhart and Zipser 1985] David E Rumelhart and David Zipser. Feature discovery by competitive learning. *Cognitive science*, 9(1):75–112, 1985.
- [Rumelhart *et al.* 1986] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.
- [Sagheer and Kotb 2019] Alaa Sagheer and Mostafa Kotb. Unsupervised pre-training of a deep lstm-based stacked autoencoder for multivariate time series forecasting problems. *Scientific reports*, 9(1):1–16, 2019.
- [Sampaio and Janeira 2003] Jaime Sampaio and Manuel Janeira. Statistical analyses of basketball team performance: understanding teams’ wins and losses according to a different index of ball possessions. *International Journal of Performance Analysis in Sport*, 3(1):40–49, 2003.
- [Sarwar *et al.* 2002] Badrul M Sarwar, George Karypis, Joseph Konstan, and John Riedl. Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In *Proceedings of the fifth international conference on computer and information technology*, volume 1, pages 291–324, 2002.
- [Schaal 1996] Stefan Schaal. Learning from demonstration. *Advances in neural information processing systems*, 9, 1996.
- [Schaal 1999] Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242, 1999.
- [Schuster and Paliwal 1997] Mike Schuster and Kuldeep K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.
- [Sephton 2016] Nicholas Sephton. *Applying Artificial Intelligence and Machine Learning Techniques to Create Varying Play Style in Artificial Game Opponents*. PhD thesis, University of York, 2016.
- [Shaffer *et al.* 2005] David Williamson Shaffer, Kurt R Squire, Richard Halverson, and James P Gee. Video games and the future of learning. *Phi delta kappan*, 87(2):105–111, 2005.
- [Shen *et al.* 2020] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of gans for semantic face editing. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9243–9252, 2020.
- [Shortliffe *et al.* 1977] Edvard H Shortliffe, W Schneider, and A-L Hein S&gvall. A rule-based approach to the generation of advice and explanations in clinical medicine. *Computational linguistics in medicine*. New York: North-Holland, pages 101–8, 1977.
- [Shute 1991] Valerie J Shute. Rose garden promises of intelligent tutoring systems: Blossom or thorn. 1991.
- [Sikorski and others 1969] Roman Sikorski *et al.* Boolean algebras. 1969.

- [Silver *et al.* 2016] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.
- [Silver *et al.* 2017] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.
- [Simonyan *et al.* 2013] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.
- [Sirlin 2006] David Sirlin. *Playing to win: Becoming the champion*. Lulu. com, 2006.
- [Spronck *et al.* 2004] Pieter Spronck, Ida Sprinkhuizen-Kuyper, and Eric Postma. Difficulty scaling of game ai. In *Proceedings of the 5th International Conference on Intelligent Games and Simulation (GAME-ON 2004)*, pages 33–37, 2004.
- [Stanley *et al.* 2005] Kenneth O Stanley, Bobby D Bryant, and Risto Miikkulainen. Real-time neuroevolution in the nero video game. *IEEE transactions on evolutionary computation*, 9(6):653–668, 2005.
- [Steinbach *et al.* 2004] Michael Steinbach, Levent Ertöz, and Vipin Kumar. The challenges of clustering high dimensional data. In *New directions in statistical physics*, pages 273–309. Springer, 2004.
- [Sundermeyer *et al.* 2014] Martin Sundermeyer, Tamer Alkhouli, Joern Wuebker, and Hermann Ney. Translation modeling with bidirectional recurrent neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 14–25, 2014.
- [Sutton and Barto 2018] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.
- [Sutton *et al.* 1999] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.
- [Svozil *et al.* 1997] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1):43–62, 1997.
- [Talwadker *et al.* 2022] Rukma Talwadker, Surajit Chakrabarty, Aditya Pareek, Tridib Mukherjee, and Deepak Saini. Cognitionnet: A collaborative neural network for play style discovery in online skill gaming platform. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3961–3969, 2022.
- [Taylor and Stone 2009] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.
- [Tealab 2018] Ahmed Tealab. Time series forecasting using artificial neural networks methodologies: A systematic review. *Future Computing and Informatics Journal*, 3(2):334–340, 2018.
- [Tekinbas and Zimmerman 2003] Katie Salen Tekinbas and Eric Zimmerman. *Rules of play: Game design fundamentals*. MIT press, 2003.

- [Tekofsky *et al.* 2013] Shoshannah Tekofsky, Pieter Spronck, Aske Plaat, Jaap van Den Herik, and Jan Broersen. Play style: Showing your age. In *2013 IEEE Conference on Computational Intelligence in Games (CIG)*, pages 1–8. IEEE, 2013.
- [Tekofsky *et al.* 2015] Shoshannah Tekofsky, Pieter Spronck, Martijn Goudbeek, Aske Plaat, and Jaap van Den Herik. Past our prime: A study of age and play style development in battlefield 3. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(3):292–303, 2015.
- [Tencé *et al.* 2010] Fabien Tencé, Cédric Buche, Pierre De Loor, and Olivier Marc. The challenge of believability in video games: Definitions, agents models and imitation learning. *arXiv preprint arXiv:1009.0451*, 2010.
- [Ter Maat 2005] Mark Ter Maat. Adaptive agents: Using flaws found in the opponents play style. 2005.
- [Thorndike 1953] Robert L Thorndike. Who belongs in the family. In *Psychometrika*. Citeseer, 1953.
- [Thue *et al.* 2007] David Thue, Vadim Bulitko, Marcia Spetch, and Eric Wasylshen. Interactive storytelling: A player modelling approach. In *AIIDE*, pages 43–48, 2007.
- [Thureau *et al.* 2004] Christian Thureau, Christian Bauckhage, and Gerhard Sagerer. Learning human-like movement behavior for computer games. In *Proc. Int. Conf. on the Simulation of Adaptive Behavior*, pages 315–323, 2004.
- [Torrey and Shavlik 2010] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.
- [Torrey *et al.* 2013] Cristen Torrey, Susan R Fussell, and Sara Kiesler. How a robot should give advice. In *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 275–282. IEEE, 2013.
- [Tucker *et al.* 2018] Aaron Tucker, Adam Gleave, and Stuart Russell. Inverse reinforcement learning for video games. *arXiv preprint arXiv:1810.10593*, 2018.
- [Tzoreff *et al.* 2018] Elad Tzoreff, Olga Kogan, and Yoni Choukroun. Deep discriminative latent space for clustering. *arXiv preprint arXiv:1805.10795*, 2018.
- [Tzu 2020] Sun Tzu. *The art of war*. Standard Ebooks, 2020.
- [Uchibe 2018] Eiji Uchibe. Model-free deep inverse reinforcement learning by logistic regression. *Neural Processing Letters*, 47(3):891–905, 2018.
- [Valls-Vargas *et al.* 2015] Josep Valls-Vargas, Santiago Ontanón, and Jichen Zhu. Exploring player trace segmentation for dynamic play style prediction. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.
- [Vaswani *et al.* 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [Vealey *et al.* 2019] Robin S Vealey, Robin Cooley, Emma Nilsson, Carly Block, and Nick Galli. Assessment and the use of questionnaires in sport psychology consulting: An analysis of practices and attitudes from 2003 to 2017. *Journal of Clinical Sport Psychology*, 13(4):505–523, 2019.
- [Vehvilainen 2001] Sanna Vehvilainen. Evaluative advice in educational counseling: The use of disagreement in the” stepwise entry” to advice. *Research on Language and Social Interaction*, 34(3):371–398, 2001.

- [Venugopalan *et al.* 2014] Subhashini Venugopalan, Huijuan Xu, Jeff Donahue, Marcus Rohrbach, Raymond Mooney, and Kate Saenko. Translating videos to natural language using deep recurrent neural networks. *arXiv preprint arXiv:1412.4729*, 2014.
- [Vesanto and Alhoniemi 2000] Juha Vesanto and Esa Alhoniemi. Clustering of the self-organizing map. *IEEE Transactions on neural networks*, 11(3):586–600, 2000.
- [Vesanto 1999] Juha Vesanto. Som-based data visualization methods. *Intelligent data analysis*, 3(2):111–126, 1999.
- [Vinyals *et al.* 2019] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.
- [Vlachos *et al.* 2002] Michail Vlachos, George Kollios, and Dimitrios Gunopulos. Discovering similar multidimensional trajectories. In *Proceedings 18th international conference on data engineering*, pages 673–684. IEEE, 2002.
- [Von Neumann and Morgenstern 2007] John Von Neumann and Oskar Morgenstern. *Theory of games and economic behavior (60th Anniversary Commemorative Edition)*. Princeton university press, 2007.
- [Wallach and Goffinet 1989] Daniel Wallach and Bruno Goffinet. Mean squared error of prediction as a criterion for evaluating and comparing system models. *Ecological modelling*, 44(3-4):299–306, 1989.
- [Wang *et al.* 2015] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.
- [Wang *et al.* 2016] Yasi Wang, Hongxun Yao, and Sicheng Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242, 2016.
- [Watkins and Dayan 1992] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.
- [Watts 2004] Duncan J Watts. *Small worlds: the dynamics of networks between order and randomness*. Princeton university press, 2004.
- [Weber and Mateas 2009] Ben G Weber and Michael Mateas. A data mining approach to strategy prediction. In *2009 IEEE Symposium on Computational Intelligence and Games*, pages 140–147. IEEE, 2009.
- [Wenger 2014] Etienne Wenger. *Artificial intelligence and tutoring systems: computational and cognitive approaches to the communication of knowledge*. Morgan Kaufmann, 2014.
- [Werbos 1990] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.
- [Williams 1971] WT Williams. Principles of clustering. *Annual Review of Ecology and Systematics*, 2(1):303–326, 1971.
- [Wolf *et al.* 2020] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers:

- State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.
- [Wu *et al.* 2021] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 22–31, 2021.
- [Xie *et al.* 2016] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, pages 478–487. PMLR, 2016.
- [Yannakakis *et al.* 2013] Georgios N Yannakakis, Pieter Spronck, Daniele Loiacono, and Elisabeth André. Player modeling. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.
- [Yarats *et al.* 2021] Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10674–10681, 2021.
- [Ye and Johnson 1995] L Richard Ye and Paul E Johnson. The impact of explanation facilities on user acceptance of expert systems advice. *Mis Quarterly*, pages 157–172, 1995.
- [Yee 2005] Nicholas Yee. Motivations of play in mmorpgs. 2005.
- [Ying *et al.* 2015] Shen Ying, Guang Xu, Chengpeng Li, and Zhengyuan Mao. Point cluster analysis using a 3d voronoi diagram with applications in point cloud segmentation. *ISPRS International Journal of Geo-Information*, 4(3):1480–1499, 2015.
- [Yuda *et al.* 2021] Kaori Yuda, Shota Kamei, Riku Tanji, Ryoya Ito, Ippo Wakana, and Maxim Mozgovoy. Identification of play styles in universal fighting engine. *arXiv preprint arXiv:2108.03599*, 2021.
- [Zhang and Sabuncu 2018] Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. *Advances in neural information processing systems*, 31, 2018.
- [Zhang *et al.* 2018] Jianming Zhang, Sarah Adel Bargal, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Stan Sclaroff. Top-down neural attention by excitation backprop. *International Journal of Computer Vision*, 126(10):1084–1102, 2018.
- [Zhuang *et al.* 2007] Yongzhen Zhuang, Lei Chen, X Sean Wang, and Jie Lian. A weighted moving average-based approach for cleaning sensor data. In *27th International Conference on Distributed Computing Systems (ICDCS’07)*, pages 38–38. IEEE, 2007.
- [Ziebart *et al.* 2008] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.
- [Zoph *et al.* 2020] Barret Zoph, Golnaz Ghiasi, Tsung-Yi Lin, Yin Cui, Hanxiao Liu, Ekin Dogus Cubuk, and Quoc Le. Rethinking pre-training and self-training. *Advances in neural information processing systems*, 33:3833–3845, 2020.

Appendix A

Raw Data

The following sections contains visualisations related to the levels and trajectories used for the Grid-World, MiniDungeons and Mariodomains.

A.1 GridWorld

The following figures are a visual representation of all the levels and trajectories generated as outlined in Section 4.5 for the GridWorld domains. This serves to demonstrate the variety in levels, both in terms of length, complexity and structure.

A.1.1 E_1

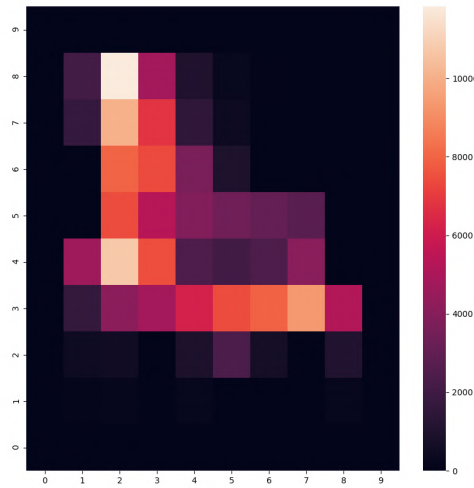


Figure A.1: Heatmap of all E_1 trajectories

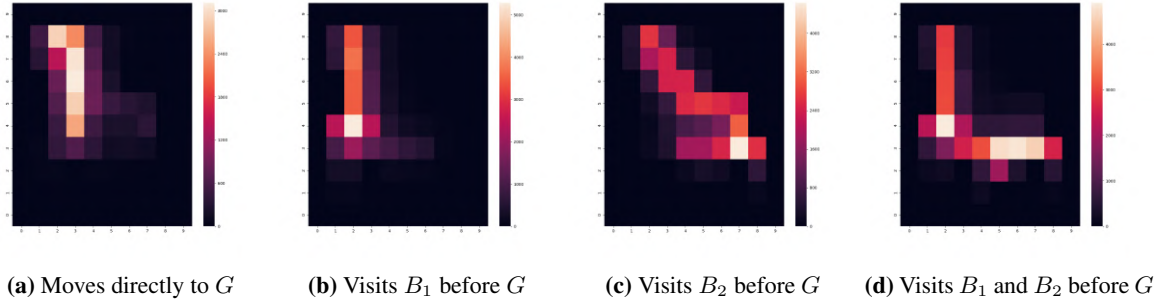


Figure A.2: Heatmap of all trajectories generated using different reward functions as described in Table 4.1

A.1.2 E_2

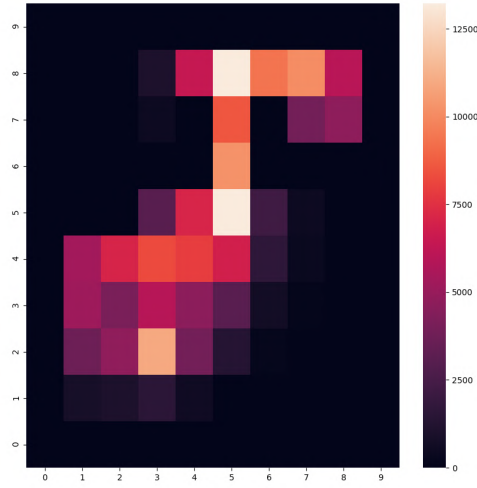


Figure A.3: Heatmap of all E_2 trajectories

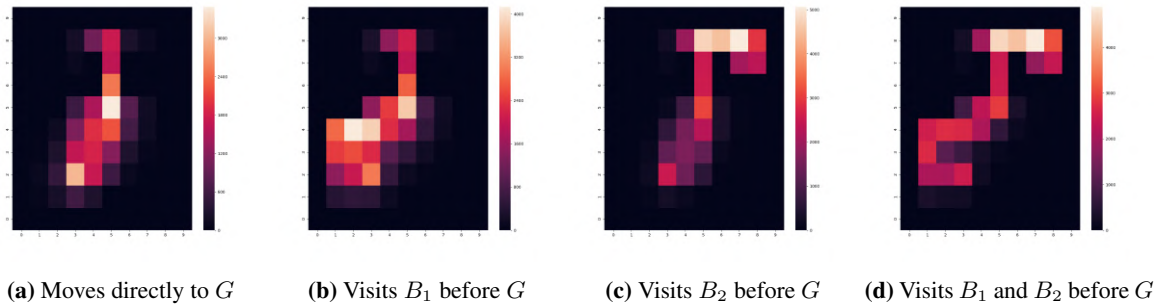


Figure A.4: Heatmap of all trajectories generated using different reward functions as described in Table 4.1

A.1.3 E_3

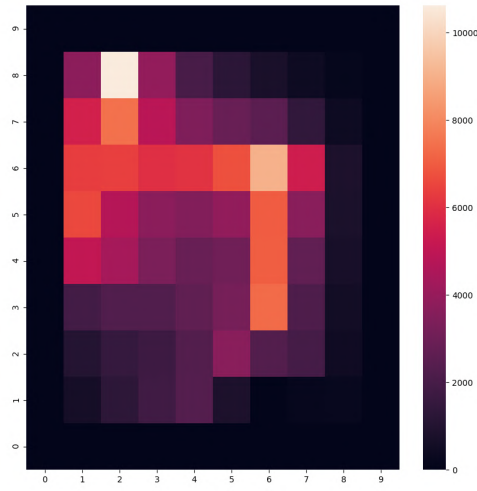


Figure A.5: Heatmap of all E_3 trajectories

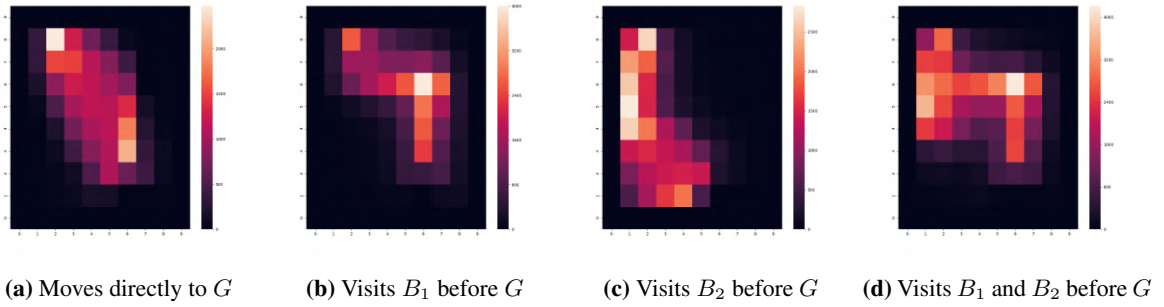


Figure A.6: Heatmap of all trajectories generated using different reward functions as described in Table 4.1

A.1.4 E_4

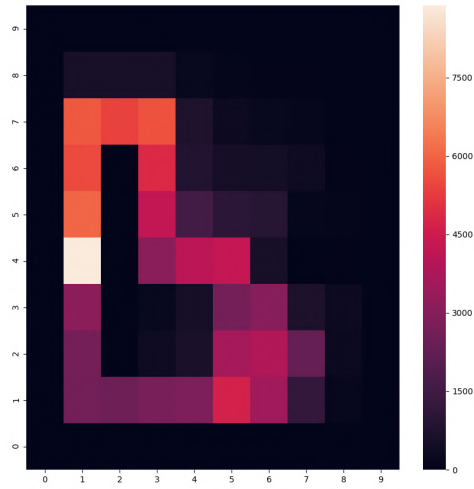


Figure A.7: Heatmap of all E_4 trajectories

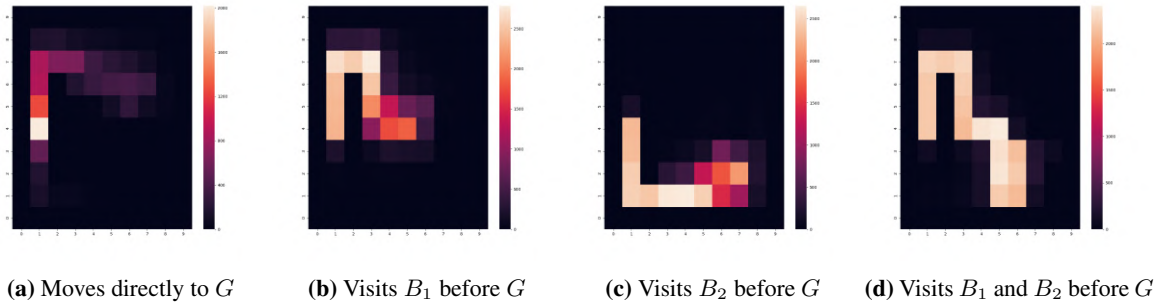


Figure A.8: Heatmap of all trajectories generated using different reward functions as described in Table 4.1

A.1.5 E_5

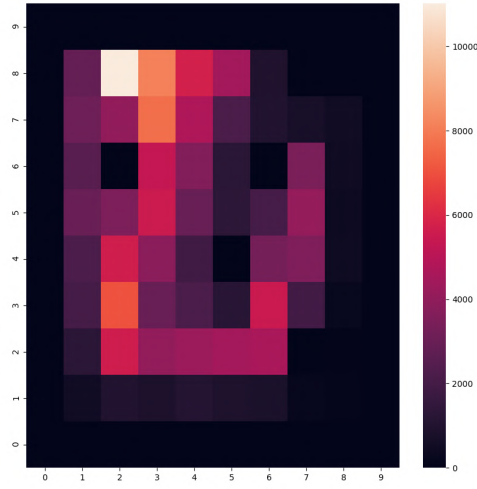


Figure A.9: Heatmap of all E_5 trajectories

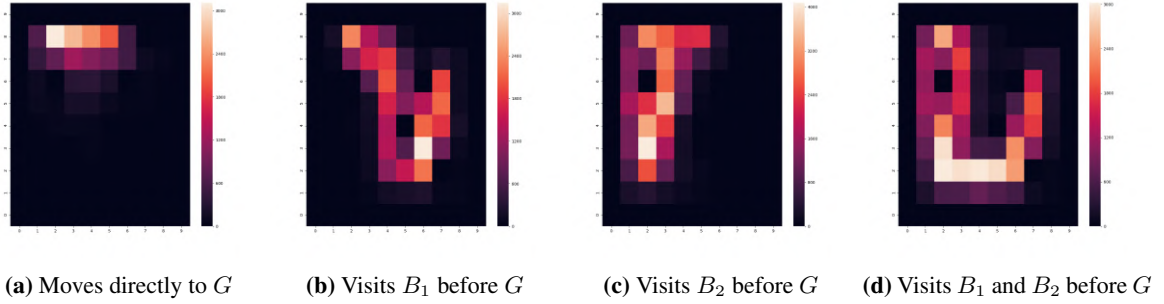


Figure A.10: Heatmap of all trajectories generated using different reward functions as described in Table 4.1

A.2 MiniDungeons

Figure A.11 is a visualisation of all the levels used by Holmgard *et al.* [2014] to generate trajectories as outlined in Section 4.5 for the MiniDungeons domain. This serves to demonstrate the variety in levels, both in terms of length, complexity and structure.

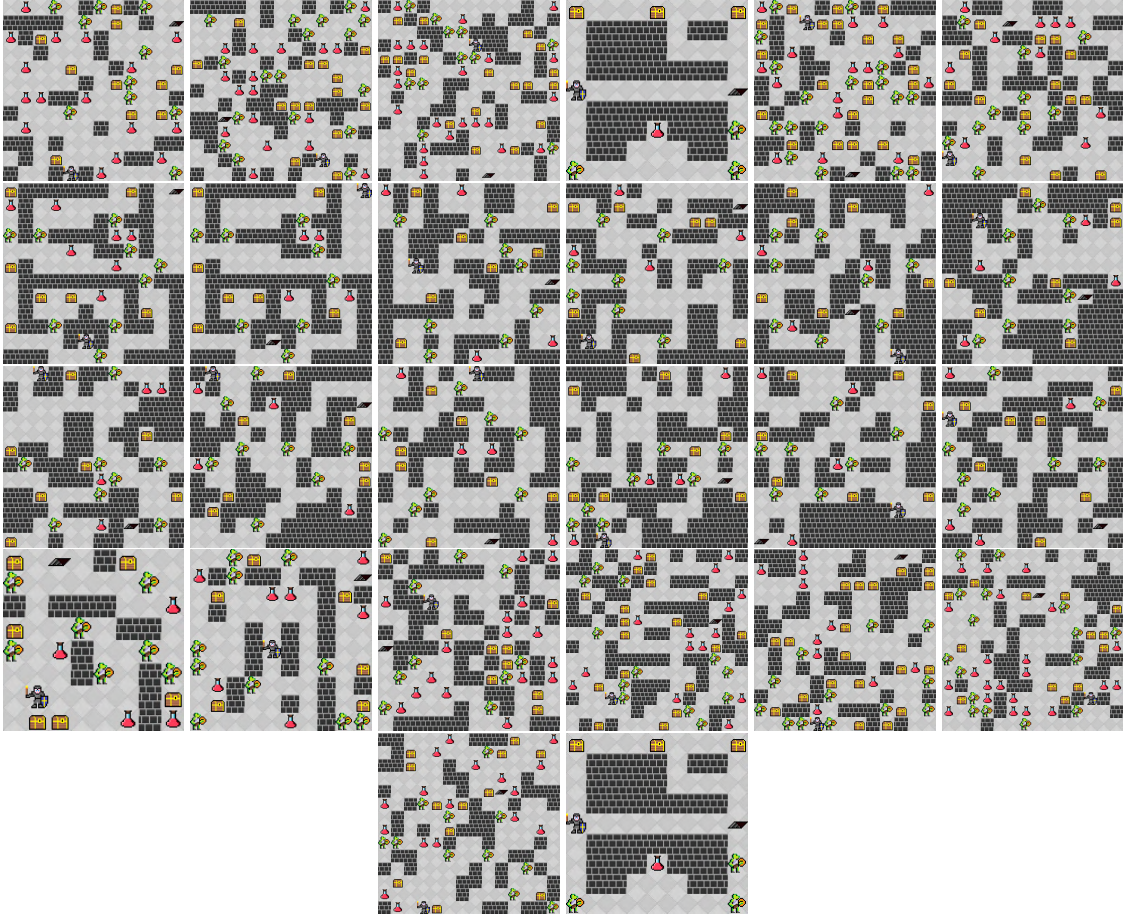


Figure A.11: Visual representation of all levels used to create Minidungeons dataset

A.3 Mario

Figure A.12 is a visualisation of all the levels used by [Guzdial and Riedl \[2016\]](#) to collect trajectories from human participants as outlined in Section 4.5 for the Mariodomain. This serves to demonstrate the variety in levels, both in terms of length, complexity and structure.



Figure A.12: Visual representation of all levels used to create Mariodataset

Appendix B

Play-style Identification

The following sections contain the visualisations of the play-style boundaries described in Section 5.2.3.3 and the play-style mean behaviours described in Section 5.2.3.4

B.1 Play-style Boundaries

Visualisation of all decision boundary regions across GridWorld domains.

B.1.1 E_1

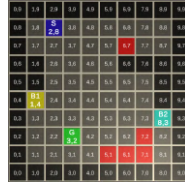
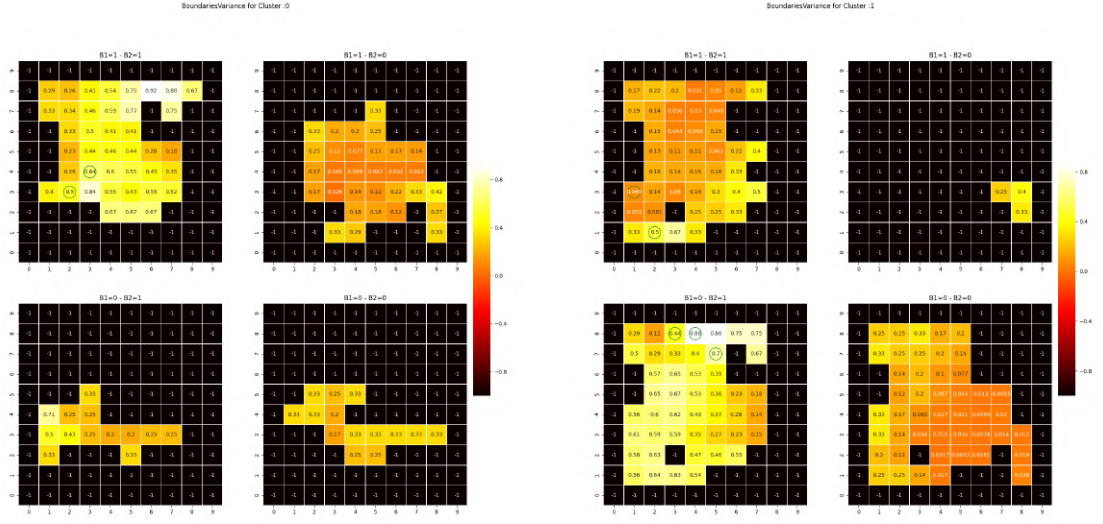
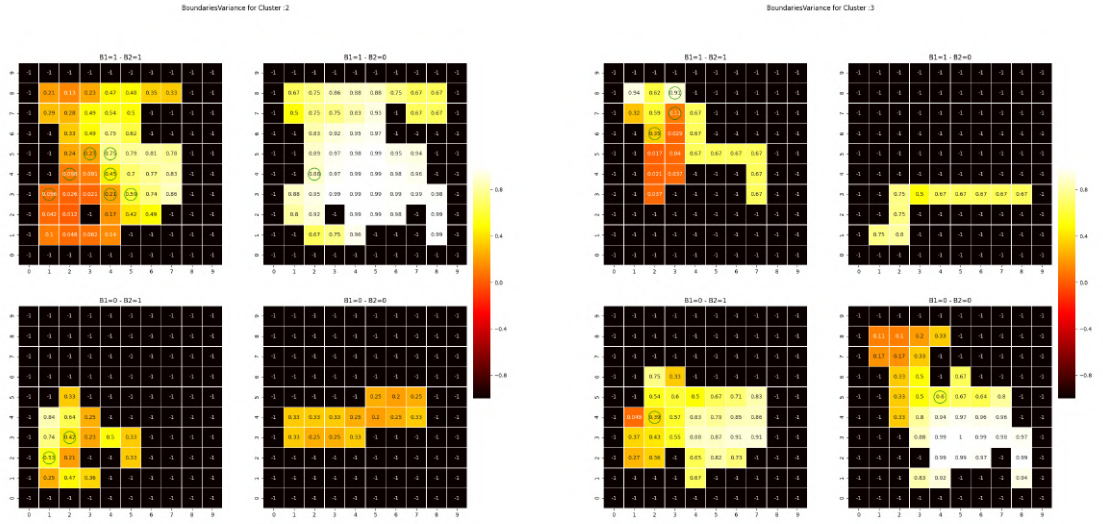


Figure B.1: Visualisation of E_1



(a) play-style cluster 0

(b) play-style cluster 2



(c) play-style cluster 3

(d) play-style cluster 4

Figure B.2: Heatmap of prediction accuracy's per-states with boundary decision boundaries depicted as green circles

B.1.2 E_2

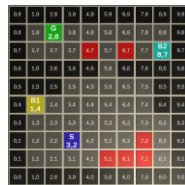
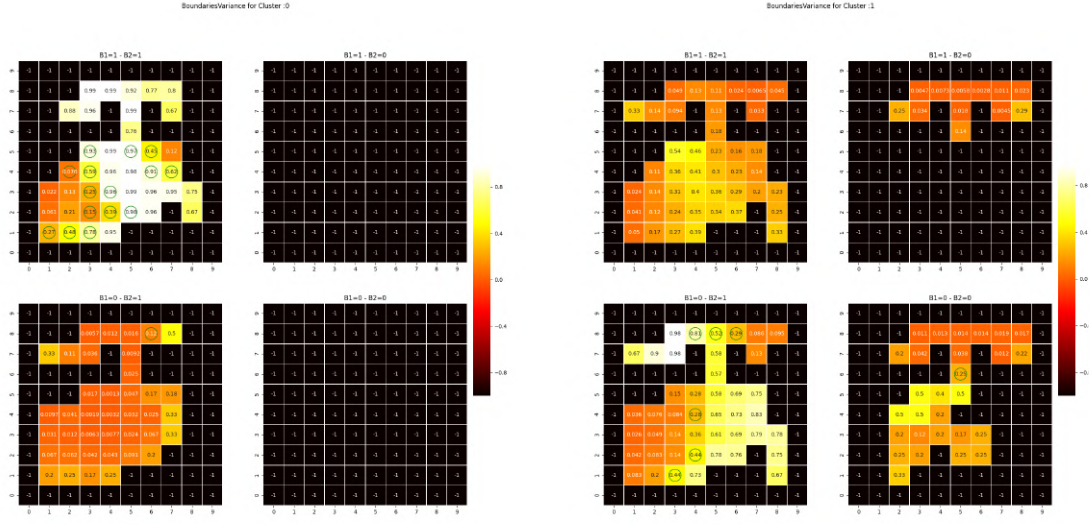
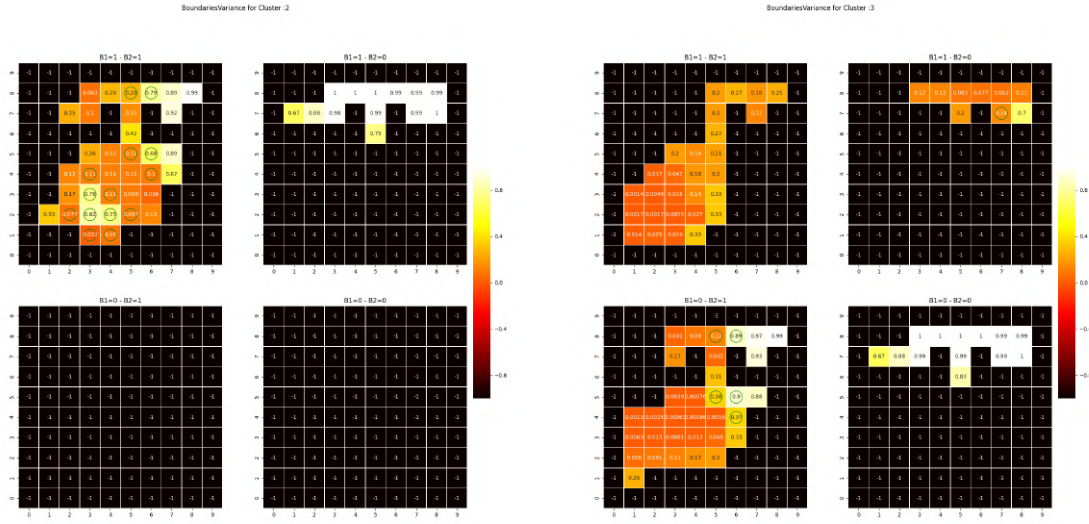


Figure B.3: Visualisation of E_2



(a) play-style cluster 0

(b) play-style cluster 2



(c) play-style cluster 3

(d) play-style cluster 4

Figure B.4: Heatmap of prediction accuracy's per-states with boundary decision boundaries depicted as green circles

B.1.3 E_3



Figure B.5: Visualisation of E_3

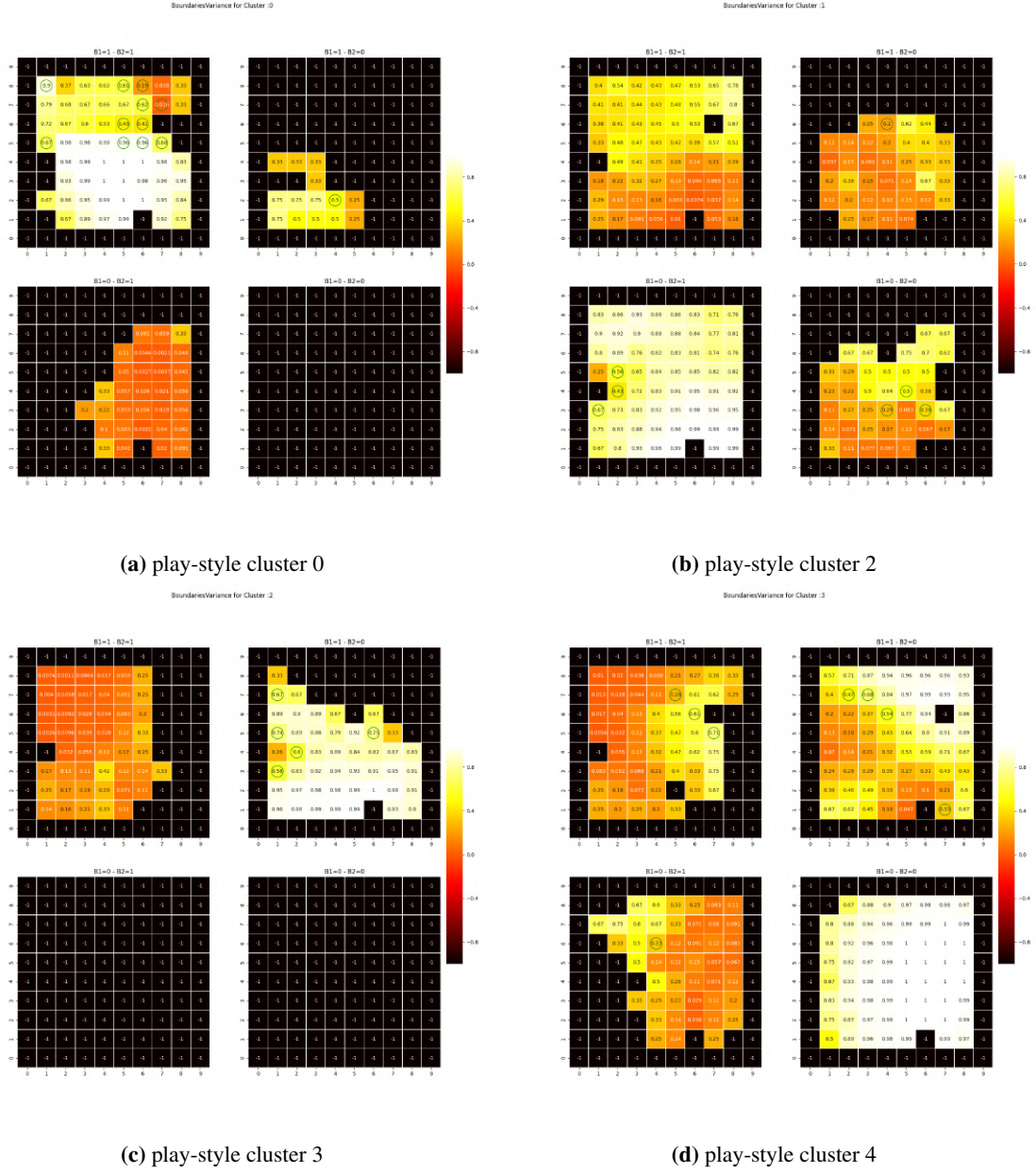


Figure B.6: Heatmap of prediction accuracy's per-states with boundary decision boundaries depicted as green circles

B.1.4 E_4

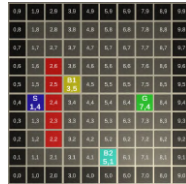
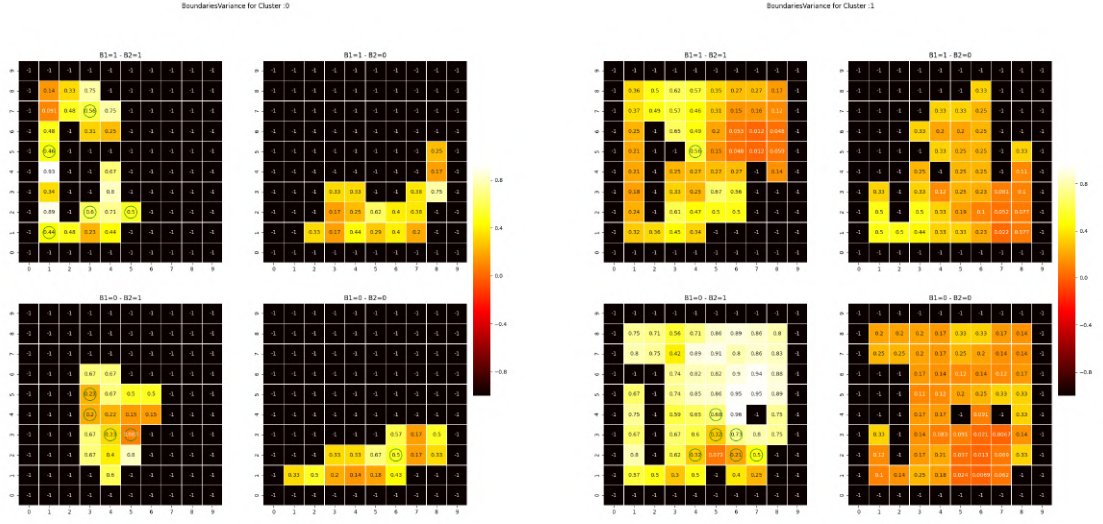
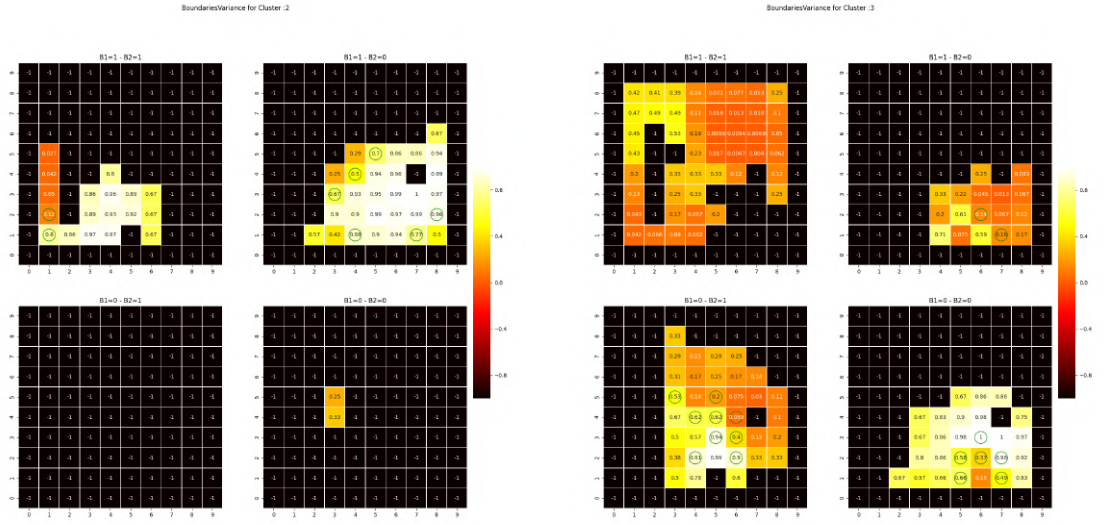


Figure B.7: Visualisation of E_4



(a) play-style cluster 0

(b) play-style cluster 2



(c) play-style cluster 3

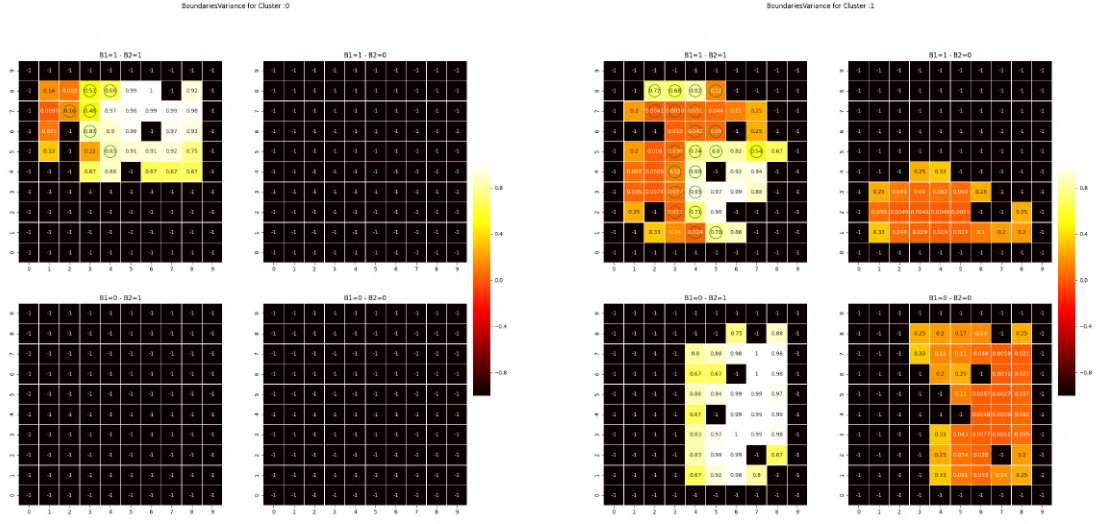
(d) play-style cluster 4

Figure B.8: Heatmap of prediction accuracy's per-states with boundary decision boundaries depicted as green circles

B.1.5 E_5

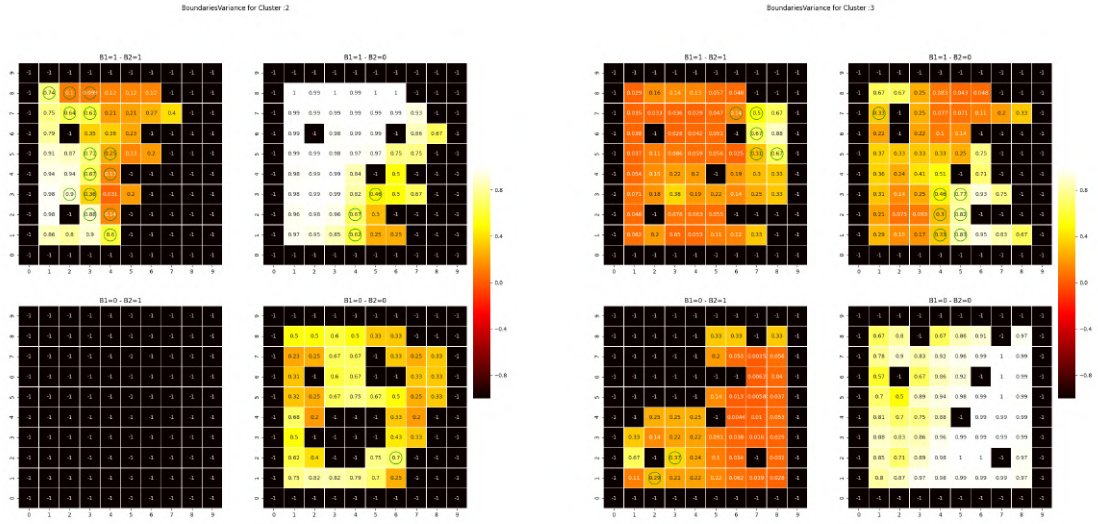


Figure B.9: Visualisation of E_5



(a) play-style cluster 0

(b) play-style cluster 2



(c) play-style cluster 3

(d) play-style cluster 4

Figure B.10: Heatmap of prediction accuracy's per-states with boundary decision boundaries depicted as green circles

B.2 Mean Behaviours

Visualisation of all mean trajectories across GridWorld domains.

B.2.1 E_1

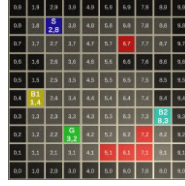
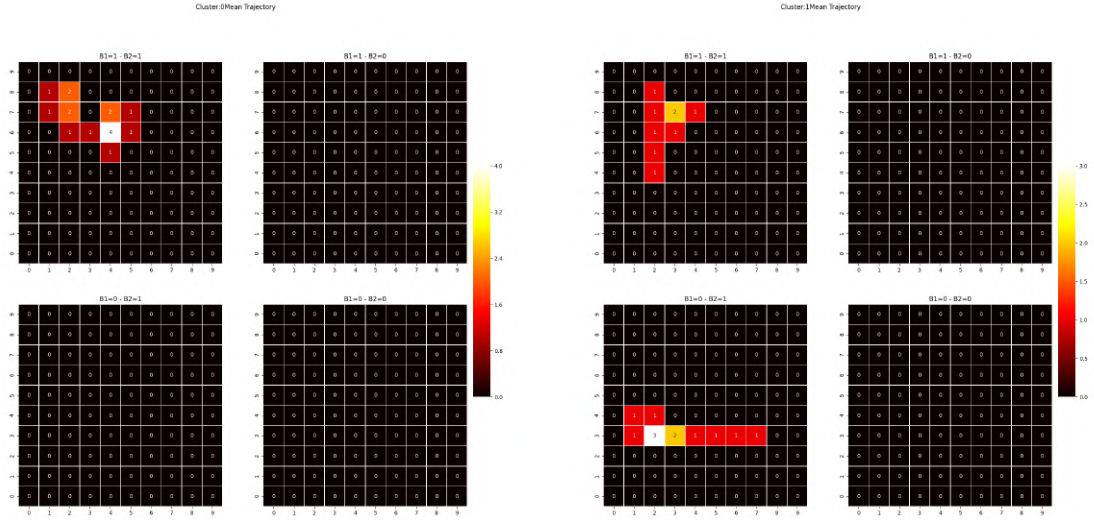
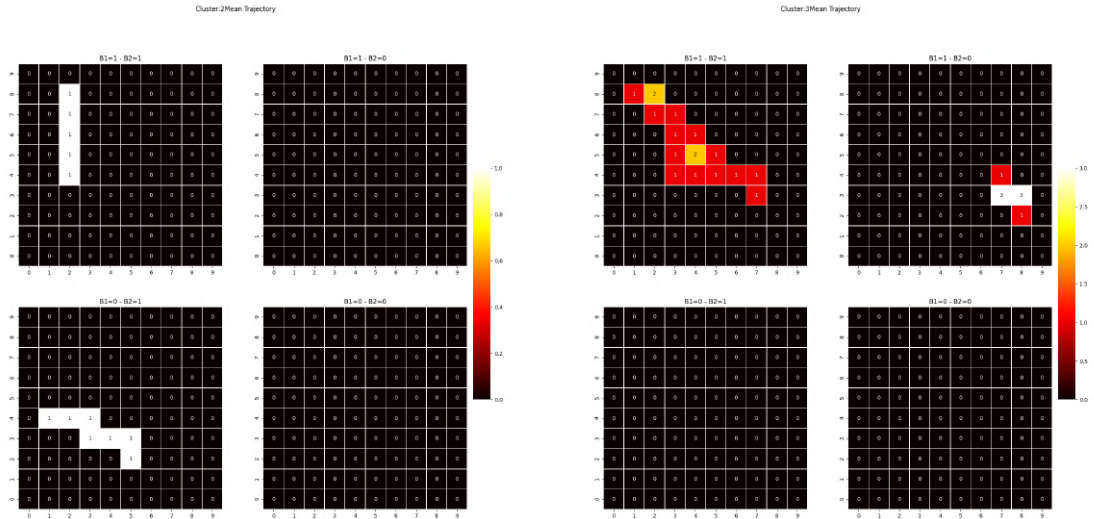


Figure B.11: Visualisation of E_1



(a) play-style cluster 0

(b) play-style cluster 2



(c) play-style cluster 3

(d) play-style cluster 4

Figure B.12: State visitation heatmap representation of mean trajectories for each play-style

B.2.2 E_2

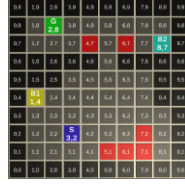
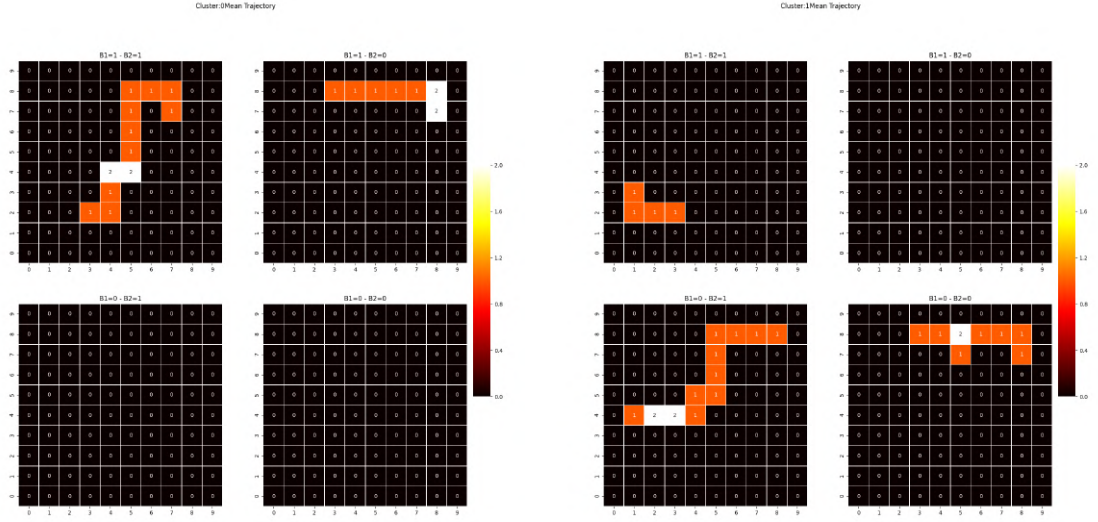
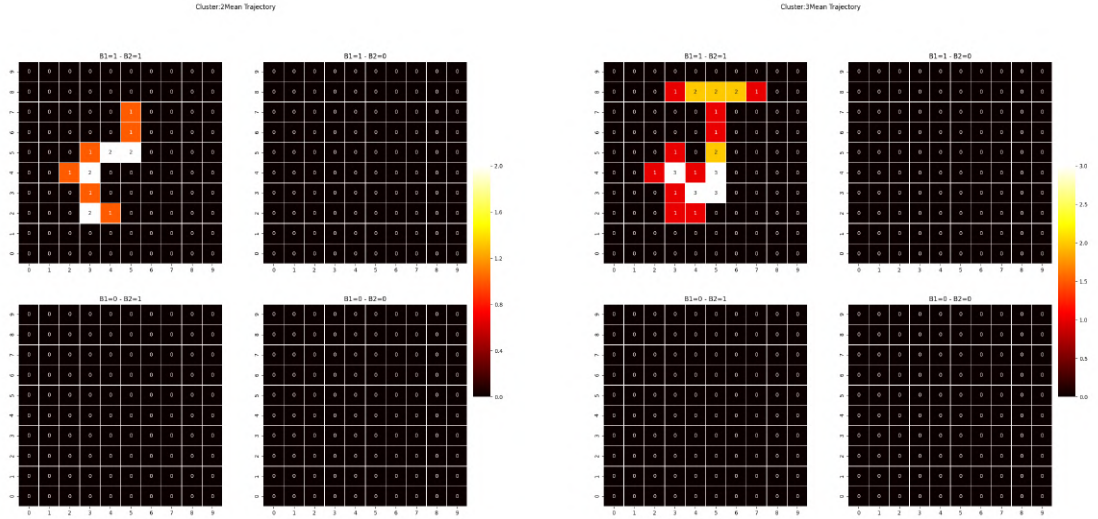


Figure B.13: Visualisation of E_2



(a) play-style cluster 0

(b) play-style cluster 2



(c) play-style cluster 3

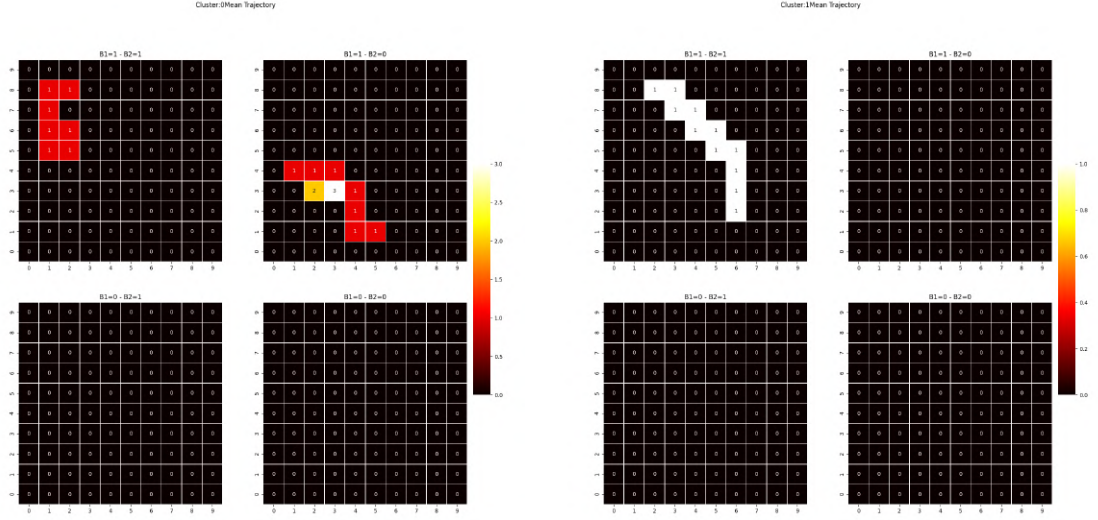
(d) play-style cluster 4

Figure B.14: State visitation heatmap representation of mean trajectories for each play-style

B.2.3 E_3

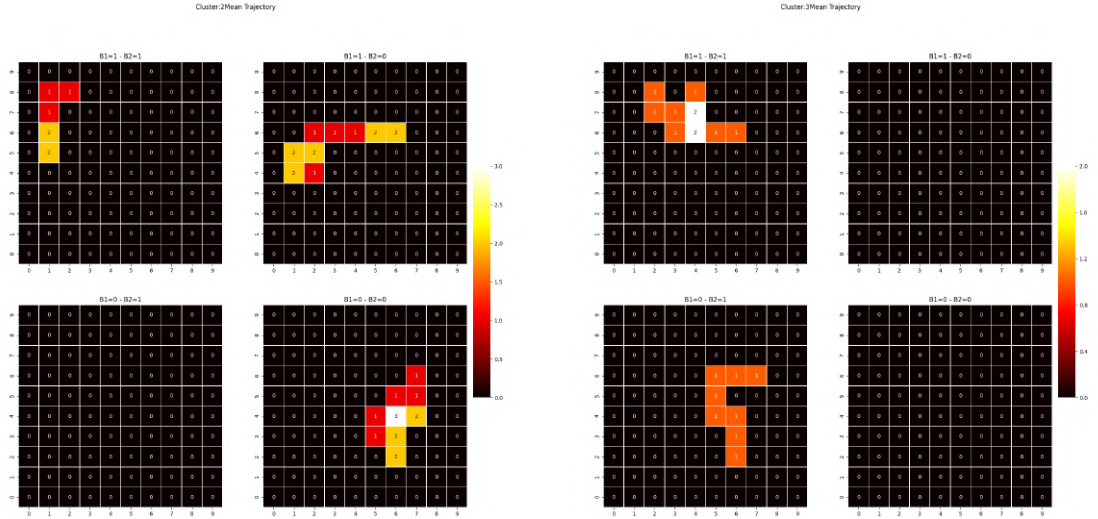


Figure B.15: Visualisation of E_3



(a) play-style cluster 0

(b) play-style cluster 2



(c) play-style cluster 3

(d) play-style cluster 4

Figure B.16: State visitation heatmap representation of mean trajectories for each play-style

B.2.4 E_4

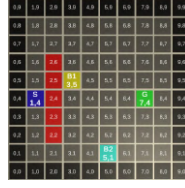
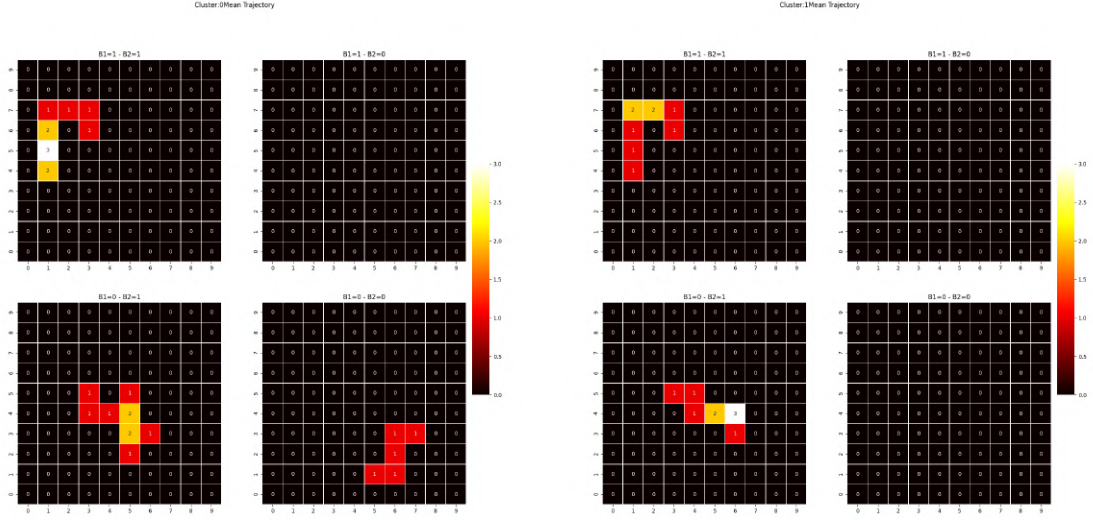
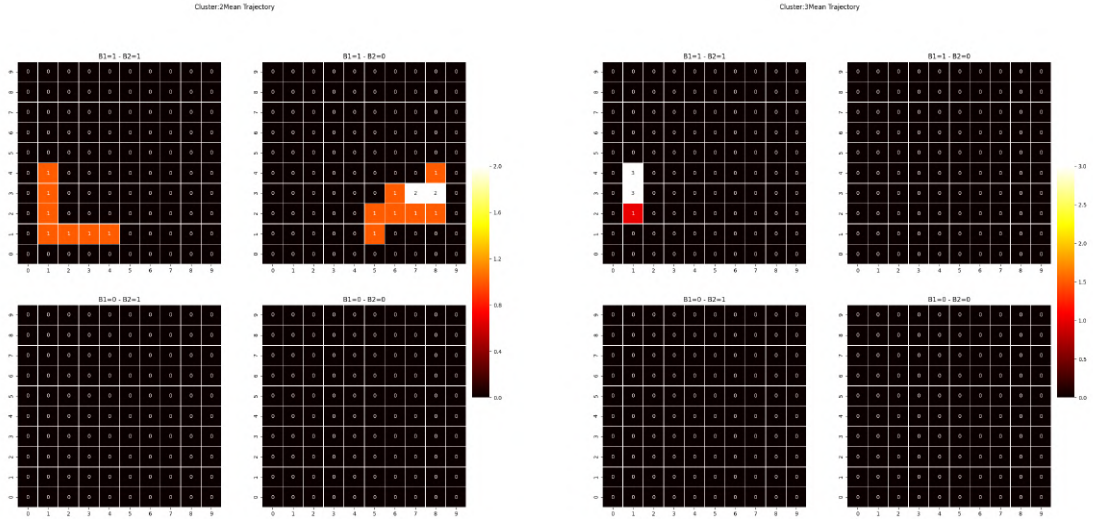


Figure B.17: Visualisation of E_4



(a) play-style cluster 0

(b) play-style cluster 2



(c) play-style cluster 3

(d) play-style cluster 4

Figure B.18: State visitation heatmap representation of mean trajectories for each play-style

B.2.5 E_5

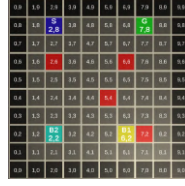
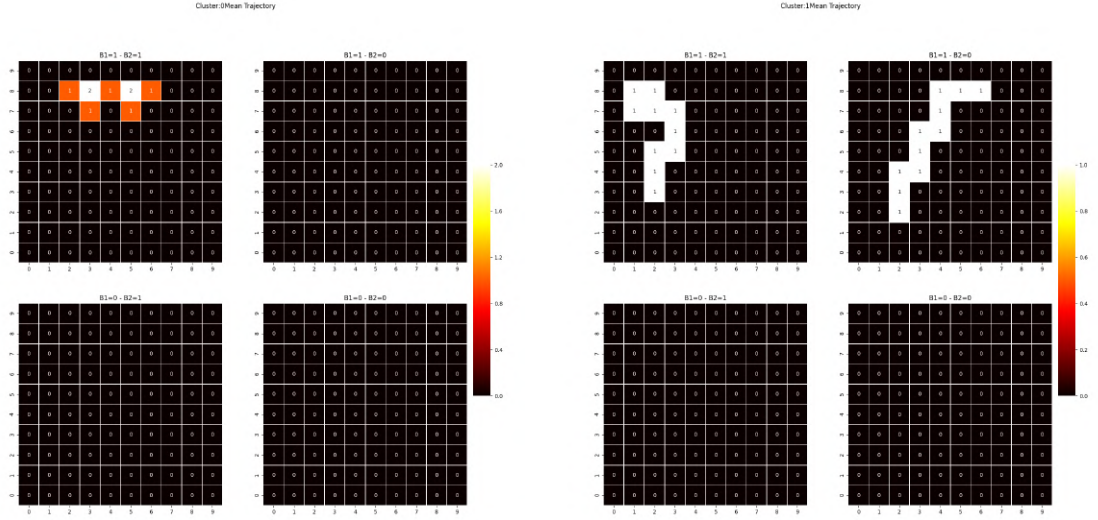
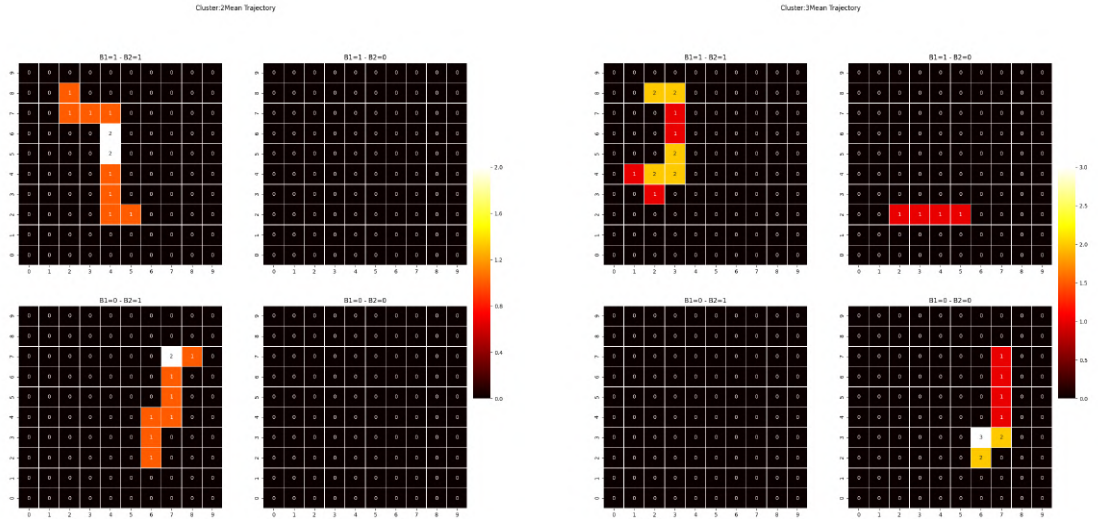


Figure B.19: Visualisation of E_5



(a) play-style cluster 0

(b) play-style cluster 2



(c) play-style cluster 3

(d) play-style cluster 4

Figure B.20: State visitation heatmap representation of mean trajectories for each play-style