Creating an Adaptive Collaborative Playstyle-Aware Companion Agent

School of Computer Science & Applied Mathematics
University of the Witwatersrand

Lindsay John Arendse
800648

Supervised by Prof. Benjamin Rosman

September 15, 2023

A dissertation submitted to the Faculty of Science, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science

Abstract

Companion characters in video games play a unique part in enriching player experience. Companion agents support the player as an ally or sidekick and would typically help the player by providing hints, resources, or even fight along-side the human player. Players often adopt a certain approach or strategy, referred to as a playstyle, whilst playing video games. Players do not only approach challenges in games differently, but also play games differently based on what they find rewarding. Companion agent characters thus have an important role to play by assisting the player in a way which aligns with their playstyle. Existing companion agent approaches fall short and adversely affect the collaborative experience when the companion agent is not able to assist the human player in a manner consistent with their playstyle. Furthermore, if the companion agent cannot assist in real time, player engagement levels are lowered since the player will need to wait for the agent to compute its action - leading to a frustrating player experience. We therefore present a framework for creating companion agents that are adaptive such that they respond in real time with actions that align with the player's playstyle. Companion agents able to do so are what we refer to as *playstyle-aware*. Creating a playstyle-aware adaptive agent firstly requires a mechanism for correctly classifying or identifying the player style, before attempting to assist the player with a given task.

We present a method which can enable the real time in-game playstyle classification of players. We contribute a hybrid probabilistic supervised learning framework, using Bayesian Inference informed by a K-Nearest Neighbours based likelihood, that is able to classify players in real time at every step within a given game level using only the latest player action or state observation. We empirically evaluate our hybrid classifier against existing work using MiniDungeons, a common benchmark game domain. We further evaluate our approach using real player data from the game Super Mario Bros. We outperform our comparative study and our results highlight the success of our framework in identifying playstyles in a complex human player setting.

The second problem we explore is the problem of assisting the identified playstyle with a suitable action. We formally define this as the 'Learning to Assist' problem, where given a set of companion agent policies, we aim to determine the policy which best complements the observed playstyle. An action is complementary such that it aligns with the goal of the playstyle. We extend MiniDungeons into a two-player game called Collaborative MiniDungeons which we use to evaluate our companion agent against several comparative baselines. The results from this experiment highlights that companion agents which are able to adapt and assist different playstyles on average bring about a greater player experience when using a playstyle specific reward function as a proxy for what the players find rewarding. In this way we present an approach for creating adaptive companion agents which are playstyle-aware and able to collaborate with players in real time.

Declaration

I, Lindsay John Arendse, hereby declare the contents of this research dissertation to be my own work. This dissertation is submitted for the degree of Master of Science in Computer Science at the University of the Witwatersrand. This work has not been submitted to any other university, or for any other degree.

Lindsay John Arendse
15 September 2023

Acknowledgements

My MSc journey has been a truly transformative experience and I am so thankful for the opportunity. A big thank you to my wife Megan who stood by my side through all the highs and lows, I achieve because of you. A special thanks to my mom Juanita, my dad Malcolm, and my brother Kyle for all the foundational support and belief in me over the many years. To the rest of my family and friends, in particular Roger, Rene, Robyn, Lauren, and Milo thank you for always being there and for reminding me that it is okay to take a break and refresh the mind.

Completing a masters part-time whilst working full-time, during a global pandemic, pushed me beyond what I thought was possible. Thank you to my previous line-managers Mergen, Dillon, Darryl, and Andy for the support and encouragement to start a masters part-time whilst working. Additional thanks to Darryl for enabling me to truly push and complete this dissertation. A wider thanks to rest of the cloud and data team for having my back during my study-leave periods. I am thankful that I got to work with such an inspiring and supportive team. Thank you to my current leadership Harry, Greg, and Merelda for the support as I worked towards my dissertation submission.

Thank you to all members of the RAIL lab for creating such a powerful and inspiring place - I'm inspired and challenged to strive beyond and further. A special thanks to Branden for the additional guidance and support, it was great collaborating with you. Thank you to my agent modelling research group (and in particular to Jarrod, Matthew, Sid, Steve, Aneesh, Iordan, and Michael) you have been apart of my research journey every step of the way. Thank you for all the ideas, insights, and constructive feedback. To my previous honours supervisor Clint, thank you for introducing and recommending me to Benji.

Finally, to my supervisor Benji, thank you so much for all the patience, guidance, and continual support. It has been an honour working with you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Companion characters in video games play a unique part in enriching player experience. In many video games such characters play a supportive role to the player [Tremblay and Verbrugge 2013]. Creating such believable and reasonable *sidekicks* poses itself as a challenging and interesting domain [Černý 2015]. A common application of Artificial Intelligence (AI) in games is to bring about intelligent behaviour in video game characters [Lou 2017]. Traditionally, game designers hand-craft agent behaviour using techniques such as Finite State Machines and Behaviour Trees. However, poor agent behaviour can be observed in situations that were not accounted for by the designer [Nguyen *et al.* 2011]. Companion agents are a specific form of in-game non-player characters (NPCs) which support the player as an ally or sidekick. Companion agents would typically help the player by providing hints, advice, resources, directions, or it may even fight along-side the human player - providing a supporting role. Companion agents are more dynamic in how they respond to the player in comparison to the more hand-scripted NPCs which exist purely to fill the game world or as enemies in the game [Bouquet *et al.* 2021].

Creating supportive companion agents is an important area of study because often the whole collaborative experience can be adversely affected if companions are not able to meet player expectation [Emmerich *et al.* 2018]. A successful companion agent should be able to overcome different kinds of collaborative problems, whilst at the same time enhance player experience [Jadhav *et al.* 2020]. Players often adopt a certain approach or strategy, referred to as a playstyle, whilst playing video games [Ingram *et al.* 2022]. Players do not only approach challenges in games differently, but also play games differently based on what they find rewarding. Companion agents should thus be designed in such a way that the game AI behaviour aligns with the player's playstyle and intention. If the companion agent is unable to predict human player intention, the human-AI collaborative experience will be less enjoyable, and thus affect the game entertainment value [Bakkes *et al.* 2012]. Human-AI collaboration, also known as team AI, is an emergent field which is predicted to grow over the coming years. Therefore, the investigation and development of techniques which enable the creation of companion agents should be pursued [Risi and Preuss 2020].

In order for companion agents to understand the behaviour of other agents, agent

modeling techniques are used [Albrecht and Stone 2018]. Researchers and developers within the field of game design and video games call this process of constructing models of agents, in their case human players, as 'player behaviour modeling' [Bakkes *et al.* 2012]. Companion agents can collaborate more effectively by using these player behaviour models to inform their action selection [Albrecht and Stone 2018]. Player behaviour modeling is of increasing importance in video games and necessary when the purpose of game AI is to improve the experience or enjoyment of the human player [Bakkes *et al.* 2012].

In this work, we explore the problem of creating an adaptive companion agent that is able to assist a human in solving collaborative games, whilst taking into account their playstyle. Creating a playstyle-aware adaptive agent firstly requires a mechanism for correctly classifying or identifying the player style, before attempting to assist the player with a given task. For this reason we divide our research problem into two research objectives:

- Objective I - *'Learn to identify playstyles'*: Create a method which can enable the real time in-game playstyle classification of players. The companion agent should be able to assist and accompany the human player throughout the game level or episode. For this reason the playstyle classification needs to occur in *real-time* at every step within the episode such that the agent can be adaptive and continue to take the most appropriate actions. If the playstyle classification process is not done at every step the resultant companion agent actions may be undesirable leading to a poorer collaborative experience since the companion agent actions depend directly on the playstyle belief during observation.

- Objective II - *'Learning to Assist'*: Create a method which makes use of the playstyle classification module, from Objective I, to assist the agent to choose the most suitable action. In other words, the companion agent should take actions which complement the player such that the player is able to achieve their playstyle specific goal. The companion agent should be adaptive such that the action policy used should be tailored and complementary to the playstyle. An action policy determines what action the agent should perform in a given situation.

The ability to correctly classify or identify the player style is a necessary requirement for the adaptive companion agent. In this dissertation we present a framework for solving this problem of in-game real time playstyle classification. We propose a hybrid probabilistic supervised learning approach, using Bayesian Inference informed by a K-Nearest Neighbours based likelihood, that is able to classify players in real time at every step within a given game level using only the latest player action or state observation. This improves on current approaches that depend instead on previous episodic player action trajectories in order to classify the player. Furthermore, we highlight the effect that this representation of the player state-action observation has on the in-game playstyle classification's accuracy, prediction stability, and generalisability. We apply and test our framework using MiniDungeons, a rogue-like dungeon exploration game, and further evaluate our framework using a natural dataset containing human player action data from the video game Super Mario Bros. The experimental results obtained from our

approach outperforms existing work in both domains. Furthermore, the evaluation results highlight the ability of our framework to generalise to unseen levels, without the need for retraining. Additionally, the Super Mario evaluation results illustrate the scalability of our framework to a more complex game environment with human player data.

One of the most important components of a collaborative companion agent is its behaviour. Companion agent behaviour that is unexpected or frustrating for the player negatively impacts the collaborative and gameplay experience [Emmerich *et al.* 2018]. The core problem which our second research objective aims to solve is this problem of determining the best, or more complementary, action response for a given playstyle. We define this problem as the 'Learning to Assist' problem: where given a set of companion agent policies, which enable the respective agents to play a given game, determine the policy which best complements the observed playstyle. In order to achieve this we create a mapping between the set of companion agents and the set of human playstyles. We represent this set of human playstyles through the implementation of agents which mimic the core characteristics of each respective playstyle. We refer to this agent implementation of each human playstyle as a proxy agent.

The mapping process between the set of companion agent policies and the set of proxy agents, as well as the companion agent policies, are computed and trained offline. We make use of Deep Q-Learning to train and develop the companion agent policies offline. The companion agent action selection step uses this 'companion policy to playstyle' mapping, companion policies, as well as the playstyle prediction output, to determine the best complimentary action. In order to evaluate our playstyle-adaptive companion agent we extend the MiniDungeons domain to a two-player collaborative game, through the inclusion of a secondary player character. In our experiment our playstyle-aware companion agent outperforms our baseline companion agents, obtaining the greatest average player-companion joint-reward across the unseen evaluation levels. The adaptive companion agent leads to a smoother player experience when using playstyle specific reward function as a proxy for what players find rewarding.

The rest of this dissertation is organised as follows: Chapter 2 discusses the related literature and ideas relevant to our research. Our research objective methodologies and experiments are respectively presented in Chapters 3 and 4. We conclude this work with Chapter 5 which recaps our research outcome, conclusion, and future work.

In summary, in this dissertation we contribute a framework for creating collaborative companion agents which are able to assist different playstyles in achieving their respective goals. As part of this framework we present a hybrid probabilistic supervised learning approach, using Bayesian Inference informed by a K-Nearest Neighbours based likelihood, that is able to classify players in real time at every step within a given game level using only the latest player action or state observation. This provides a solution to the problem of in-game real time playstyle classification. As part of our playstyle classifier evaluation we introduce a measure called the playstyle prediction stability. The prediction stability is a measure of how often a playstyle classifier changes its playstyle prediction class. Prediction stability is a necessary feature, not present in related stud-

ies, for real time playstyle classification since frequent fluctuations in the prediction can adversely affect the game's adaptability and personalisation. Furthermore, we successfully apply our real time playstyle classification method through the creation of our collaborative companion agent. During our work we have contributed[1] documentation and several other bug-fixes and enhancements to the open source MiniDungeons project, and as a result have been made a project maintainer by the owner. Additionally, we contribute Collaborative MiniDungeons, our two-player extension to MiniDungeons which as a research testbed enables future work within the field of multi-agent research.

Our work on real time in-game playstyle classification, Research Objective I, has been accepted for publication and appears as part of the Third Southern African Conference for AI Research Proceedings, featuring within the Communications in Computer and Information Science (CCIS) Springer book series [Arendse et al. 2022].

Arendse, L.J., Ingram, B., Rosman, B. (2022). Real Time In-Game Playstyle Classification Using a Hybrid Probabilistic Supervised Learning Approach. In: Pillay, A., Jembere, E., Gerber, A. (eds) Artificial Intelligence Research. SACAIR 2022. Communications in Computer and Information Science, vol 1734. Springer, Cham. https://doi.org/10.1007/978-3-031-22321-1_5

[1] https://github.com/ganyariya/gym-md/graphs/contributors

Chapter 2

Background

The research field of artificial intelligence (AI) in games, namely game AI, first emerged in the 1950s and today the application of AI is seen throughout the game industry [Xia *et al.* 2020]. Our work is centered around companion agent creation and falls within the intersection of the following game AI subfields [Xia *et al.* 2020]:

- Player behaviour modeling, which encompasses techniques focused on creating a model approximation of player actions and behaviour in order to assist in understanding player goals [Xia *et al.* 2020][Albrecht and Stone 2018][Bakkes *et al.* 2012]. Player behaviour is related to actions taken by the player in-game [Yannakakis and Togelius 2018]. See Section 2.1 for a discussion on Player modeling.
- Game AI for believable non-player characters (NPCs), which is centered around the creation of intelligent game playing agents that can bring about a more immersive or challenging experience for players [Yannakakis and Togelius 2015].
- Team AI, an emergent field which focuses on AI-human and AI-AI agent team collaboration [Risi and Preuss 2020].

There is a close mutualistic connection between these fields, such that human player models can inform and help design believable agent architectures, and vice versa [Yannakakis and Togelius 2015]. The creation of believable behaviours for non-player characters is key to improving player experience during game play [Cruz and Uresti 2018]. Autonomous agents can bring about a better player experience if the agent can understand player goals and actions. Autonomous agents can better understand player goals and actions through agent modeling techniques [Albrecht and Stone 2018]. From a video game design perspective this process of constructing models of other agents is known as 'player behaviour modeling' [Bakkes *et al.* 2012]. According to Risi and Preuss [2020] the field of Team AI is expected to grow over the coming years. We begin our discussion with Section 2.1 which discusses work related to player behavioural modeling. In Section 2.2 we discuss the work relevant to playstyle classification. We then discuss the literature related to companion agent creation in Section 2.3.

2.1 Player behavioural modeling

Yannakakis *et al.* [2013] define player modeling as the study of computational mod-

els of players in games. Such computation models aim to represent the player's behavioural, cognitive, and affected states, all of which are based on data (or theories) informed from the in-game interaction of the human player and the game. From the perspective of player behavioural modeling, a player model is an abstracted representation of a player's behaviour in a game environment [Bakkes *et al.* 2012].

Player modeling provides a mechanism which game designers and researchers can use to gain insight and understanding into how players are feeling and how players might act [Zhang and Verbrugge 2018]. Such insight may result in a more enjoyable, or a smoother, player gameplay experience. Player models can also be used to inform companion agents of possible player actions in order to respond appropriately [Tremblay and Verbrugge 2013] (this kind of application of player models is also known as *adaptive player experience* or *game balancing*). Player models has been used in bringing about adaptive procedural content generation [Scirea 2020]. It is important to note that player modeling techniques can also be applied to other in-game non-player agents [Bakkes *et al.* 2012]. Furthermore, the modeling of non-player agents allow for the testing of player modeling techniques.

Game environments present a number of challenges which player modeling techniques need to overcome. Often the categorisation of players into an existing player model needs to happen in near real time, whilst other computations, such as graphics rendering and game physics, occur simultaneously [Yannakakis *et al.* 2013]. Therefore the techniques used in player modeling should strive to be computationally efficient [Bakkes *et al.* 2012].

There are two high level approaches to player modeling: model-based, also called top-down, and model-free, also called bottom-up. Model-based approaches use theoretical frameworks that theorise models to explain player behaviour. Model-free approaches are driven by data collected during player-game interaction. Model-free approaches refer to the creation of a model which maps player input (player actions and behaviour) and player state (representation of player experience) [Yannakakis and Togelius 2018].

Bakkes *et al.* [2012] define four model-free categories for player behaviour modeling: (1) modeling player actions, (2) modeling player tactics, (3) modeling player strategies, (4) profiling a player. Action models are focused on modeling the in-game actions that players take. An action model is usually made up of a list of possible game states, paired with possible actions and a probability value which represents the likelihood of the player taking a specific action in the given state. In other words, an action model can be viewed as a function that takes a set of actions within a state and chooses the appropriate action a player would likely perform. A tactic can be viewed as the player 'motives for actions taken'. Player tactic modeling is focused on modeling a player's local short-term (sub-goal) behaviour, which is made of or can be described as a sequence of player actions taken. Modeling player strategies is centered around modeling longer-term (global goal) behaviours, which are accomplished through a sequence of successful tactics. Finally, player profiling is concerned with the creation of psychological and social-based player profiles which aim to explain player observed behaviour

and player internal traits such as personality and preference. Player profiling is often closely related to player experience (or user) modeling.

In video games bringing about an adaptive or personalised experience requires a mechanism for correctly classifying or identifying the player style, before attempting to modify the experience in some way which improves player interest and immersion. Section 2.2 discusses related work central to the problem of playstyle classification in video games.

2.2 Playstyle Classification Related Work

In video games there are sometimes several *ways* or *styles* which players use to play a game. Different players find different parts of a game challenging and rewarding. Catering for this diversity in player playstyle, preference, and skill is quite challenging for game creators [Charles and Black 2004]. Adapting to player preference and skill is important in achieving higher levels of player engagement and is especially vital in games used for education [Bontchev 2016]. Furthermore, from a game design perspective, in order for game developers to maximise the number of target players it is essential to create games which cater for different gameplay experiences or playstyles [Tychsen and Canossa 2008].

A player playstyle can be seen as a representation of the player's strategy and player profile [Bakkes *et al.* 2012]. In order to understand or react accordingly to a given playstyle a model approximation of the player is needed. Approaches within this area of player behaviour modeling are centered around the creation of this model approximation of players [Yannakakis *et al.* 2013]. In other words, a player model can be defined as an abstracted representation of a player's behaviour in a game environment [Bakkes *et al.* 2012]. Player modeling provides a mechanism which game designers and researchers can use to gain insight and understanding into how players are feeling and how players might act [Zhang and Verbrugge 2018]. Player modeling in itself is an interesting challenge. Player models are additionally useful when combined with game personalisation or game adaptability (also known as *dynamic difficulty adjustment* [Gonzalez-Duque *et al.* 2021], *adaptive player experience* or *game balancing* [Tremblay and Verbrugge 2013]). This application of player models to game personalisation or adaptability is of increasing importance in video games and is especially necessary when the purpose of game AI is to improve the experience or enjoyment of the human player [Bakkes *et al.* 2012].

There are a number of closely related areas of work centered around the problem of creating adaptive or personalised games. Several studies focus on the player model creation process, in other words, modeling or mimicking human players. Holmgård *et al.* [2021] model human players through the creation of computation personas which can be used for automated testing within games. This work illustrates that computation personas, or human behavioural clones, can acceptably represent human player playstyles [Holmgård *et al.* 2014c]. The human behavioural clones are able to cap-

ture human playstyles when compared to hand-designed clones created through expert knowledge. Furthermore, the work by Holmgård *et al.* [2014a] shows that human proxy agents created based on the main objectives within a given game can express the distinct behaviour seen in different human playstyles. Iwasaki and Hasebe [2022] present a framework for generating playstyles within a given game without predefining or hand-crafting a reward function. Their approach is able to evolve and generate a diverse set of human player behaviour through the clustering of gameplay log data, also referred to as play logs.

Several researchers focused on how the game experience can be changed or personalised to the identified player. Game personalisation that is able to cater for different player preference is an important challenge [Gonzalez-Duque *et al.* 2021]. Gonzalez-Duque *et al.* present a method, based on Bayesian Optimization, which is able to dynamically adapt the difficulty of a game in response to the player's skill level. The difficulty of the game is adjusted such that the game is not too easy or too challenging for the player. The work by Togelius *et al.* [2006] also aims to maximise player engagement through the automated generation of tailor-made game levels within a racing game. They use an evolutionary algorithmic approach to evolve the respective racing tracks in response to a player model based on pre-collected player log data recorded at each time step. The player modeling approaches used were not sufficient to capture the true driving style of human players. However, the player models are acceptable for the main purpose of responding and adapting racing levels to each player's model behaviour. The work by Ingram [2021] aims to address this personalisation problem within tutorial-based learning systems. Tutorial based learning systems, like video games, need to cater for diverse user preferences (or learning styles). Ingram discusses a method which can identify different player types, train optimal agents which mimic these identified styles, and respond with best-advice. The problem not fully explored is how often should the playstyle identification be done in order to tailor the experience best and in turn how often should advise be given. Creating an adaptive player experience which is enjoyable is a challenging and subjective task. Furthermore, the problem of recovering and identifying the underlying playstyles present in play logs still remains and requires further work. Additional work is needed to explore the impact different play log representations have on the playstyle prediction performance.

Whilst adapting the player experience is important, once first needs a mechanism for identifying, classifying, or clustering the player. There are several researchers trying to answer this question of 'does the observed player belong to a known player type or playstyle?' Normoyle and Jensen [2021] investigates a method which uses Bayesian semi-parametric clustering for creating player clusters. They take post-match data and cluster based on how the player's choices affect the end game result, in contrast to clustering on the outcomes directly. Iwasaki and Hasebe [2022] use a clustering approach, called x-means, to cluster play log data with the aim of evolving different agent playstyles created using a genetic algorithm, also called C-NEAT [Iawasaki and Hasebe 2021]. Past work in this field has been largely concerned with identifying playstyles at the end of the game episode, in order to evolve or create diverse player persona agents, or to analyse player trace data for analytical or game testing purposes.

The user-study work by Valls-Vargas *et al.* [2015] present a player modeling framework to capture non-stationarity within player strategy by using sequential machine learning techniques which incorporate predictions from previous temporal player observations. However, the effect of the playstyle classifiers' prediction stability is not a component of their approach. Prediction stability is a necessary feature for real time playstyle classification since frequent fluctuations in the playstyle prediction can adversely affect the game's adaptability and personalisation. Additionally, Valls-Vargas *et al.* [2015] relied on a manual annotation process for providing playstyle labels to the collected player trace data at fixed intervals. This manual annotation is not only time consuming but may also be infeasible from a cost perspective. Thus there is a need for further work exploring alternative feasible solutions to this lack of labelled player datasets. Hernandez-Leal *et al.* [2016] also considered the concept of non-stationary strategies. They utilised a Bayesian framework to train an agent which learns an optimal policy against an opponent whose strategy switches.

The work by Ingram *et al.* [2022] looks to remedy the issue of requiring labelled player data by utilising an unsupervised LSTM autoencoder clustering approach. Their autoencoder model works by projecting trajectory information into a lower dimensional latent representation which can then be clustered using vanilla clustering approaches like Gaussian Mixture Models. The resultant clusters are interpreted as different playstyles, since similar trajectories will be grouped together. This autoencoder trajectory projection into a lower latent space is how the authors address the problem of processing variable length player action trajectories. Since different playstyles can produce different episodic lengths, which results in a difference in the length of the player action trajectory data. This problem of processing variable length player action trajectories is important to overcome. Ingram *et al.* apply an unsupervised LSTM-autoencoder clustering approach to the problem of playstyle identification. The work by Ingram *et al.* presents itself as a suitable comparative study since their playstyle identification model is able to identify playstyles during gameplay using player action trajectories. One limitation is that the clustering step in their approach performs better when previous episodic player action trajectories is included during the playstyle prediction process. In other words, their approach is dependent on previous episodic player action trajectories. This means their model cannot acceptably predict a playstyle using only the latest state and observed action. A playstyle classifier should ideally be able to accurately predict a playstyle from the start of an episode and be less reliant on needing to first observe the player in order to collect enough data for accurate prediction. Furthermore, the authors explore an unsupervised learning approach since access to labeled player playstyle data is hard to obtain. There is still a need for further work regarding approaches or user-studies which address this lack of labelled player action data.

Gow *et al.* [2012] utilise a clustering approach which incorporates multi-class Linear Discriminant Analysis (LDA) to model players in Snakeotron and Rogue Trooper. Similar to Valls-Vargas *et al.* [2015] they rely on in-game event occurrences and summaries. These in-game events only occur after a sequence of actions or after a certain amount of time. Such a delay in player behavioural feedback will limit the game's adaptability

and prevent playstyle classification from occurring in real time, and therefore prevent a truly responsive experience.

It is necessary that the classification of players has to happen in real time and concurrently with other computations, such as graphics rendering, non-player character behaviour logic, and game physics [Yannakakis *et al.* 2013]. Therefore techniques used in real time playstyle identification should strive to be computationally efficient [Bakkes *et al.* 2012].

Computationally complex classifiers pose a risk at runtime by interrupting or halting the game experience. The user study work by Scott and Khosmood [2017] highlights this adverse effect which computationally heavy approaches have on a game's playability and player experience. Scott and Khosmood conclude that approaches which reduces the playability of the game and thus the overall player experience are unacceptable in a commercial video-game setting. Furthermore, from a model explainability perspective understanding the playstyle prediction outcome, identifying which game mechanics or features certain playstyles use, and gaining insight into how the different player type clusters relate are useful to game designers when deciding on which game mechanics to change, and which new game features to implement or remove [Ingram *et al.* 2022][Tychsen and Canossa 2008]. For this reason, explainable machine learning approaches should be utilised.

2.3 Related work on companion agents

A companion agent can be defined as in-game characters which support the player as an ally or sidekick. An ally or sidekick would usually help the player by providing hints, advice, resources, directions, or it may even fight along-side the human player [Bouquet *et al.* 2021]. The inclusion of a companion agent into a game is not trivial and brings about a number of challenges and considerations. Agent behaviour is one of the most important components of a collaborative companion agent. Unexpected agent behaviour can adversely affect the collaborative experience. There are a number of approaches to bringing about agent behaviour. These game AI approaches can fall into non-adaptive game AI (e.g. finite state machines and scripting) or adaptive game AI that is often realised through machine learning techniques [Emmerich *et al.* 2018].

There are often numerous ways or styles in which players can play a game. These player styles, or playstyles, can be seen as a representation of the player's strategy and player profile [Bakkes *et al.* 2012]. Respective playstyles can be identified and defined in various ways. One way as we have seen is through clustering techniques [Kline and Arlidge 2003]. Other approaches involve game data analytics to predict player behaviour [Yannakakis and Togelius 2018] [Mahlmann *et al.* 2010]. Ideally we would desire companion agents to take into account player playstyle. If companion agents can recognise and assist accordingly to a respective playstyle, the collaborative and game experience may be improved [Freedman and Zilberstein 2018].

The work by Zhang and Verbrugge [2018] make use of player modeling techniques to model human player understanding of game AI movements. Tremblay and Verbrugge [2013] investigate the problem of companion agents within first person shooter genre of games. The approach makes use of a player behaviour model to inform companion agent actions. The companion agents' adaptive behaviour is driven by several behaviour trees which meet different playstyle expectations (cautious, support, or aggressive play). The learning behaviour tree approach, by Tomai and Flores [2014], makes use of in-game observed player behaviour to dynamically adapt the agent behaviour tree. Nguyen *et al.* [2011] present the *CAPIR: Collaborative Action Planning with Intention Recognition* framework, which uses concepts from decision theory and planning techniques to create agents that are able to assist a human in collaborative games. In their work, they assume that the human cooperator requires the help of the agent to solve an unknown task. The CAPIR framework makes use of planning techniques and is based on solving problems which can be modelled as a Markov Decision Process (MDP). Please see Section 2.4 for a discussion of the Markov Decision Process.

Solving MDPs that have large state-spaces and many variables is challenging. The *POMCoP: Partially Observable Monte-Carlo Cooperative Planning* framework by Macindoe *et al.* [2012] like the CAPIR framework, can be used for creating companion agents which are able to reason about the player's intended goal and cooperate based on this. The difference is that the POMCoP framework models the game world as a partially observable Markov decision process (POMDP), whilst the CAPIR framework uses an MDP. This allows for agents built using the POMCoP framework to plan in the belief space rather than the state space. Planning in the belief space allows the agent to reason about how the actions taken can affect the uncertainty about the player intention. From a scaling perspective the approaches by Nguyen *et al.* [2011] and Macindoe *et al.* [2012] can be computationally taxing and struggle to scale to larger domains that require a reasonable model of human behaviour [Černý 2015]. The work by Cruz and Uresti [2018] propose a hierarchical reinforcement learning framework that can be used for creating believable agents that appear to be controlled by a human player.

An important observation highlighted in the work, by Valls-Vargas *et al.* [2015], is that existing work on player modeling often assume that the playstyle of players is static. In some cases, players change their playstyle as time passes (i.e. there is a 'temporal' change in player behaviour). Valls-Vargas *et al.* propose a player modeling framework which addresses the problem of inferring a player's dynamic playstyle. The work by Hernandez-Leal *et al.* [2016] is focused on the problem of recognizing different agent strategies and propose a Bayesian framework to address this. Holmgård *et al.* [2014c] investigate the problem of modeling human playstyles in order to evolve agents approximating human playstyles (more specifically, decision making styles) using the MiniDungeons player modelling game domain to test and benchmark their approaches. Ghosh *et al.* [2020] highlight and study the problem of designing agents that can collaborate with other agents of unknown types in situations that are multi-agent, also known as the 'zero-shot coordination' problem [Hu *et al.* 2020]. The approach by Ghosh *et al.* models the zero-shot coordination problem with a parametric MDP and relies on the agent first observing the human player's actions in order to infer the human

player's type and then accordingly adapt the policy to collaborate effectively.

The work by Emmerich *et al.* [2018] provides an analysis of companion agent characteristics, as well as a discussion on how companion agents affect player experience and expectations. One of the big challenges faced when creating companion agents is enabling the agents such that they can understand and comprehend their cooperator's intention. In addition, the agent needs to decide the best way to assist based on the intention as well as game state [Nguyen *et al.* 2011]. Contemporary games still struggle to convey interesting and effective AI controlled companion agents. Often players would rather prefer to collaborate with an unskilled human over an AI controlled sidekick [Černý 2015]. The work by Černý explores to what extent can this player choice be attributed to poor quality of AI and to what extent is this a game design problem. Furthermore, the work by Černý discusses the problems, for companion agents, that will remain even if agent competence level can be made satisfactory. The problems discussed are: human-agent communication, team friendship, and how the agent is perceived which align with the challenges related to agent appearance, social stance, and communication that are highlighted in the work by Emmerich *et al.* [2018]. The human-study work by Friedman and Schrum [2019] also aims to understand player preference for cooperative hand-coded agents over less cooperative but more skilled agents, that use neuroevolution. Friedman and Schrum found that the player agent preference is dependent on what the human player values (winning vs. team based tactics). When an agent is not adaptive to different players and when the agent is unable to align to what the player values then the the collaborate performance of the human-agent team can be adversely effected [Hu *et al.* 2020][Ghosh *et al.* 2020].

2.4 The Markov Decision Process and Deep Q-Learning

A Markov Decision Process (MDP) is a mathematical framework, often used in reinforcement learning (RL), to model the internal decision making processes, dynamics, and outcomes within a system or environment, which may be stochastic or deterministic. A MDP consists of [Sutton and Barto 2018]:

- A set S of states, which contains an initial state s_0.
- A set A of actions
- A state transition function $P : S \times A \times S \to [0, 1]$.
  The function $P(s, a, s')$ outputs the probability [0,1] of the agent "transitioning" from state s to state s' using an action a, where $a \in A$ and $s, s' \in S$.
- A reward function $R : S \times A \times S \to \mathbb{R}$.
  The function $R(s, a, s')$ outputs the reward an agent receives given that the agent transitioned from state s to state s' using action a, where $a \in A$ and $s, s' \in S$.

An important assumption made with MDPs and within RL is the Markov property. The Markov property states that if an agent is in state s and takes an action a at time step t, then the resultant state s' at time step $t + 1$ depends only on state s and action a. In

other words, any states and actions taken, by the agent, prior to state s and action a at time step t should have no impact or effect on the state s' at time step $t+1$. The Markov property should also hold true for the reward function: meaning the reward value returned for the resultant state s' at time step $t+1$ should only depend on the action a at time t in state s [Marsland 2014][Russell and Norvig 2010][Sutton and Barto 2018].

A MDP may be *continuous* meaning that there is no end goal state; the episode does not terminate. A MDP may also be *episodic* which means the MDP contains a terminal or end goal state, i.e. the epsiode terminates. We restrict this discussion to episodic MDPs and thus define an *episode* as each combination of states and actions that take the agent from the start state to the end goal state [Marsland 2014][Russell and Norvig 2010].

A *policy* is any function π, such that $\pi : S \rightarrow A$, which will return an action $a \in A$ given any state $s \in S$. Policies can be deterministic or probabilistic. A *policy* can be seen as a *solution* specifying what action an agent should take in any given state [Marsland 2014][Russell and Norvig 2010].

The agent uses its policy to determine the action to take from the start of the episode to the end of the episode. For a given episode a *return* will be yielded based off the actions taken and the rewards received. The *return* of an episode, under a policy π, is the sum of all the rewards received: $R^{\pi}(s_0) = r_1 + r_2 + ... + r_T = \sum_{i=1}^{T} r_i$, where $s_0 \in S$ and $r_1, r_2, ..., r_i \in \mathbb{R}$ and where T is the final time-step in which the agent reached the goal state [Marsland 2014][Russell and Norvig 2010].

In order to learn and exploit the MDP environment, actions which maximise the expected return should be taken. In a given state the expected reward is referred to as the *value* of the state. The *value* of a state $s \in S$, with respect to a policy π can be defined as the expected return if we start in state s and generate an episode using the policy π: $V^{\pi}(s) = \mathbb{E}(R^{\pi}(s))$. A policy π is an optimal policy, denoted as π^*, if $V^{\pi^*}(s) \geq V^{\pi}(s), \forall s \in S$ and $\forall \pi$. Q-learning is one approach to finding an optimal policy through the use of a Q-function. The Q-function is used to estimate the *value* for a given state [Russell and Norvig 2010].

The Q-function, with respect to a policy π, is defined as: $Q^{\pi} : S \times A \rightarrow \mathbb{R}$. The function $Q^{\pi}(s, a)$, where $a \in A$ and $s \in S$, is the expected return for the agent starting in state s and taking action a first, and then following the rest of the episode defined by the policy π. The Q-learning algorithm makes use of a *Q-table* to inform the values of the Q-function [Sutton and Barto 2018].

The *Q-table* is a table that stores the $Q(s, a)$ value for all $s \in S$ and $a \in A$. Each time the agent is transitioned into a new state the value of $Q(s, a)$ is updated, in the Q-table, using the resulting state and reward, as shown in Algorithm 1.

Algorithm 1 Q-learning Algorithm

Require: $num\_of\_episodes$, num_of_steps, $\alpha \in (0, 1]$ *is the learning rate,* $\gamma \in (0, 1]$ *is the discount rate,* $\epsilon \in [0, 1]$ *is a probability value*

1: Initialiation: set $Q(s, a)$ to small random values for all $s \in S$ and $a \in A$
2: **for** episode in $num\_of\_episodes$ **do**
3: $s \leftarrow$ env.initialise()
4: **for** step in num_of_steps **do**
5: select action a using the ϵ-greedy policy
6: take action a and receive reward r
7: sample new state $s^{'}$
8: update $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s^{'}, a^{'}) - Q(s, a)]$
9: set $s \leftarrow s'$
10: **end for**
11: **end for**

In each new episode the agent makes use of the Q-table to decide on which actions are best in which states. However, with a small probability value ϵ the agent will not take the action dictated by the Q-table value but rather a random action. This randomness allows for new state-action combinations to be explored. Therefore, after a large number of episodes the policy determined by the Q-values in the Q-table is defined as:

$$\pi(s) = \begin{cases} argmax \;\; Q(s, a), & with\ probability\ 1 - \epsilon \\ random \;\; action \;\; a, & with\ probability\ \epsilon \end{cases}$$

where $s \in S$, $a \in A$, and $0 \leq \epsilon \leq 1$. This policy is commonly known as ϵ-greedy [Sutton and Barto 2018].

In Line 8 of Algorithm 1 the Q-value update rule is:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s^{'}, a^{'}) - Q(s, a)] \tag{2.1}$$

The $Q(s, a)$ update is not just influenced by the received reward r, but also by the difference between the terms $\max_{a'} Q(s^{'}, a^{'})$ and $Q(s, a)$. The term $\max_{a'} Q(s^{'}, a^{'})$ is the maximum Q-value of the possible resultant state-action pairs. In Deep Q-learning this is called the target values. This difference between the target Q-values and the output Q-values is referred to as the loss.

The creation process of the Q-table does not scale well to complex MDP environments which have large state and action spaces - since the tabular Q-table size is directly proportional to the number of states and actions. Therefore, instead of using the Q-table to inform the values returned by the Q-function. Neural networks can be used as a function approximation, for this Q-function, to return the Q-values for the given state and action pairs. Neural networks that are applied in this way are referred to as a Q-networks. In order to approximate the Q-values in larger high-dimensional complex state and action spaces, the Q-network is made up of many neural layers and is thus called a deep Q-network (DQN) [Kaelbling *et al.* 1996].

The input layer of the DQN takes in the state s and outputs the Q-value estimates for the possible actions. In Deep Q-learning we are using backpropagation to adjust the DQN weights in order to minimise the loss between the output Q-values and the target Q-values. For a given input state the DQN outputs the Q-values for the possible actions and is called the policy network. In other words, the policy network outputs the $Q(s, a)$ values. In Deep Q-learning there is no Q-table to look up the target $\max_{a'} Q(s^{'}, a^{'})$, therefore a secondary target network is to give us the target values. A loss function such as the mean squared error (MSE) is typically used to update the policy network according to the difference between the output Q-values and the target Q-values.

The target network is a copy of the policy network, but the weights are updated only after a certain number of steps. This is done to address the moving target problem. The weights in the target network, which approximates the optimal Q-value in the resultant state $s^{'}$, should be updated less frequently since this is what is used to calculate the loss and optimise the policy network. If the target network weights are updated concurrently with the policy network, then during the loss calculation the target Q-values would change every step - leading to longer training times [Sutton and Barto 2018].

In Deep Q-learning, the states are not passed sequentially to the policy network as done in Q-learning. Instead, the states, actions, the resulting reward, and resulting state are collected and stored in a data structure called replay memory. The replay memory stores the past experiences of the agent's policy network and is randomly sampled from. This is done in order to break the correlation between subsequent states [Mnih and et al. 2015]. The training process of the DQN is outlined in Algorithm 2.

Algorithm 2 High-level training process for Deep-Q network with respect to Q-learning

Require: $memory\_size$, $memory\_batch\_size$, $random\_seed$, $num\_of\_episodes$, num_of_steps, $learning\_rate$, $discount\_rate$, $epsilon$

replay_memory ← initialise($memory\_size$)
policy_network ← initialise_weights($random\_seed$)
target_network ← clone_weights($policy\_network$)
for episode in $num\_of\_episodes$ or done is True **do**
    s ← env.initialise()
    for step in num_of_steps **do**
        action ← select_action(s, $epsilon$) *# epsilon: exploration vs. exploitation*
        s', reward, done ← env.step($action$)
        replay_memory.store(s, $action$, $reward$, s')
        q_values ← policy_network(replay_memory.sample($memory\_batch\_size$))
        target_q_values ← target_network(using s' resulting from q_values)
        loss ← calculate_loss(q_values, $target\_q\_values$) *# e.g. mean squared error*
        perform_backpropagation($policy\_network$, $loss$)
        update_weights($policy\_network$, $optimiser$)
        # optimiser algorithm, can be stochastic gradient descent (SGD) or a variant like Adaptive Moment Estimation (Adam)
        if target_network update condition is met **then**
            target_network ← clone_weights($policy\_network$)
        end if
    end for
end for

Chapter 3

A Framework for an Adaptive Collaborative Playstyle-Aware Companion Agent

In this dissertation, we explore the problem of creating an adaptive playstyle aware companion agent that is able to assist a human in playing collaborative games. A playstyle-aware adaptive agent able to confidently assist the player in real time requires two capabilities: a mechanism for correctly classifying or identifying the player style and secondly the agent needs to respond with an expected or complementary action. We present our framework in two parts: Section 3.1 presents a method for playstyle classification and Section 3.2 presents a method for developing the adaptive agent.

3.1 Playstyle Classification

The ability to correctly classify or identify the player style is a necessary requirement for the adaptive companion agent. In this section we present our method for solving this problem of in-game real time playstyle classification. Furthermore, we present a formalisation of the Playstyle In-game Classification Problem in Subsection 3.1.1. As illustrated in Figure 3.1 our framework consists of multiple steps where low-level player action trajectory data are processed to generate corresponding labeled playstyle play log data which is used by the playstyle classifier creation process. Our framework has two parts: Player Action Data Processing and Playstyle Classification.

The Player Action Data Processing outlined in Subsection 3.1.2 is focused on how the player action play log data is generated. In Subsection 3.1.3, which outlines the Playstyle Classification process, we present our hybrid probabilistic supervised learning approach, using Bayesian Inference informed by a K-Nearest Neighbours based likelihood, that is able to classify players in real time at every step within a given game level using only the latest player action or state observation. Our framework improves on current approaches that depend on previous episodic player action trajectories in order to classify the player.

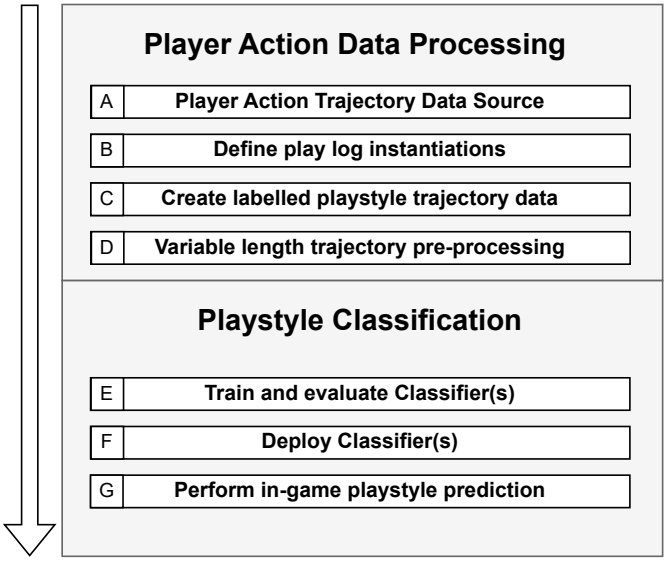


Figure 3.1: Proposed Framework Overview

3.1.1 Background for the problem of in-game playstyle classification

This section describes the terminology and definitions necessary for defining the in-game playstyle classification problem.

Play Log Definition

Game metrics are the recorded numeric data generated during gameplay as a result of the player interacting with the game [Tychsen and Canossa 2008]. Playstyles in games are identified based on the observed in-game event information [Iwasaki and Hasebe 2022]. This is a reason why the use of a *play log* [Iawasaki and Hasebe 2021], which captures such event information, is valuable for identify play styles. Thus our classification approaches uses this concept of a play log, which is a vector containing the essential game metrics required to identify or discriminate between playstyles [Iawasaki and Hasebe 2021] [Iwasaki and Hasebe 2022].

We define the play log as pl, an n-by-k dimensional vector containing $n \times k$ game metrics m:

$$pl = \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,k} \\ m_{2,1} & m_{2,2} & \cdots & m_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n,1} & m_{n,2} & \cdots & m_{n,k} \end{bmatrix}$$

where $n, k \in \mathbb{N}$ and $m \in \mathbb{R}$. In order to fully describe the playstyle classification problem and for notation, we introduce Υ and Ψ. Let Υ be an instance of pl, with a specific n, k, and m values. Let Ψ be an instance of pl containing unseen play log game metric data with the same n, k, and m values as Υ. Υ represents the historic seen play log data for a given game and is used for the training of our classifier, whereas

Ψ represents the set of play log data which is not known, or seen beforehand. Ψ can be thought of as *live* play log data or data that is not used during the training of the playstyle classifier. Ψ is the play log data that will be generated during the evaluation of our playstyle classifier as different playstyle behaviours interact with the hold-out testing set of game levels. It is important to note that the game metrics are recorded and updated at every step within a given level of a game.

Playstyle Set

We make use of proxy agents, also called personas [Holmgård *et al.* 2014a][Holmgård *et al.* 2014c], to mimic human playstyles within the given game and thus generate the play log metric data. For a given game g, there may exist a number of possible playstyles. We call this set of all possible playstyles for a given game g: P_g. Since we may not be able to implement agents which cover all the possible playstyles in P_g, we define a subset called Γ, where $\Gamma \subseteq P_g$. Γ represents the set of playstyles which will be implemented using a proxy agent.

Game Levels

It is common for games to have one or more levels. Thus for a given game, divide all available levels into a set for training. The training level set is used to train the playstyle classifier. The remaining levels should be used for evaluating the playstyle classifier.

In-game Playstyle Classification Problem Definition

We aim to create a classifier that is trained on labeled play log data, Υ, called Ω_Υ. The labeled play log data, Υ, is generated using the set of proxy agents and the set of training levels from a given game g. The model classifier Ω_Υ should be able to classify play log entries, Ψ, to a given playstyle in Γ.

$$\Omega_\Upsilon : \Psi \rightarrow \Gamma \tag{3.1}$$

The classifier Ω_Υ should, at every step within a given game level, classify the observed agent, using the play log data, to a playstyle in Γ. The model classifier Ω_Υ is evaluated using the respective evaluation levels. The general logic behind the playstyle classifier prediction and evaluation process is shown in Algorithm 3. Our hybrid classifier is discussed in Subsection 3.1.3.

3.1.2 Player Action Data Processing

Our playstyle identification framework is able to account for two cases related to the availability of player action play log data, as shown in Figure 3.2: a general case where no play log data exists, and the special case where player action play log data exists. The general case commonly occurs since the availability of player action data is usually not available during the early stages of game development. The special case is less common since human player action data is hard to obtain or is often not made public by game publishers.

Algorithm 3 Playstyle Classifier Prediction and Evaluation Process

Require: $test\_mode$, exp_levels, Γ, Ω_Υ
$num\_episodes$, $episode\_length$

```
agents ← get_agents(Γ)
for level in exp_levels do
  for playstyle_agent in agents do
    for episode in num_episodes do
      env ← initialise(level)
      done = false
      classifier ← load(Ω_Υ)
      for step in episode_length do
        action ← playstyle_agent.action(env)
        env, playlog_entry, done ← env.step(action)
        prediction ← classifier.classify(playlog_entry)
        record_accuracy(prediction, classifier, playstyle_agent, level)
        if done is true then
          break
        end if
      end for
    end for
  end for
end for
```

Our framework begins with the *Player Action Trajectory Data Source* step (Figure 3.1(A)). Our aim is to build playstyle classifiers which make use of play log metric data to distinguish between different playstyles. During the early stages of game development, especially when mechanics of a game are being tested, changed, and developed it is challenging for game designers to collect game metric data from human players [Iwasaki and Hasebe 2022]. Furthermore, playtesting with human players can be time consuming and a costly exercise. This is why quite a number of studies create and make use of agents as a proxy for human players [Holmgård *et al.* 2021][Holmgård *et al.* 2014c][Iwasaki and Hasebe 2022]. Hence, for the general case we need to generate synthetic player action trajectory data using these human behavioural agents.

A domain specific design decision is the play log instantiations used within the given game (Figure 3.1(B)). As discussed in Subsection 3.1.1 the play log contains the essential game metrics required to identify or discriminate between playstyles. For this reason the construction of the play log is important and several play log representations should be considered.

Once the play log representations has been defined, the next step in our framework is to, in both the general and special cases, obtain labelled playstyle trajectory data (Figure 3.1(C)). In the general case, proxy agents which mimic human player behaviour are used to generate the play log data. The proxy agent playstyle used to generate this

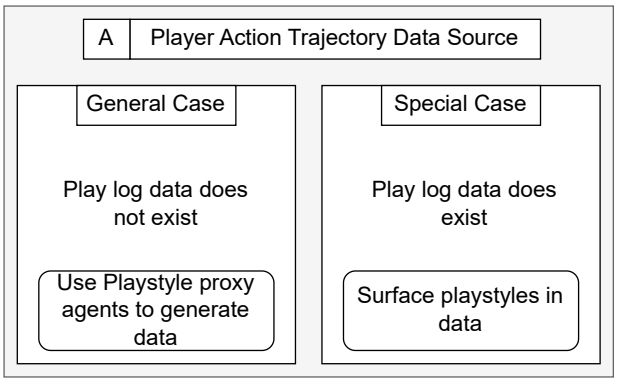


Figure 3.2: Player Action Trajectory Data Source, Step (A) from Figure 3.1

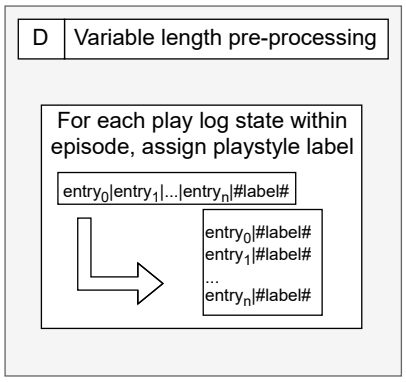


Figure 3.3: Variable length trajectory pre-processing, Step (D) from Figure 3.1

play log data is the corresponding playstyle label. In this way labelled playstyle play log data is obtained.

For the special case when existing play log data exists, traditional unsupervised clustering is used to obtain the playstyle label for the respective play log data. The special case occurs when real human player play log data is made available; as a result of user studies or when commercial game publishers make their player gameplay data available.

To handle the variable length of player action trajectories across different episodes, we pivot the episode play log such that each individual entry within the play log is assigned the play log playstyle label for the episode (Figure 3.1(D)). For example, consider the following play log obtained at the end of an episode with the respective playstyle label called *ps_label*:

```
14,3,0,3,2,2|14,3,0,3,3,24|14,3,0,3,3,7|14,3,0,3,3,22|#ps_label#
```

Each individual play log entry, from the resultant step in the episode, is seprated by a |. Each index in the individual play log entry corresponds to a metric within a given game (this can be a score, player health, or numbers of items collected). The playstyle label, *ps_label*, is delimited by the surrounding # symbols. As mentioned, in order to handle the variable length of different episodes, we pivot the episode play log such that each individual play log entry within the episode play log is assigned the play log playstyle label of the episode. Using the above episode play log example the resultant play log looks as follows after the *pivot* and playstyle label assignment:

```
14,3,0,3,2,2|#ps_label#
14,3,0,3,3,24|#ps_label#
14,3,0,3,3,7|#ps_label#
14,3,0,3,3,22|#ps_label#
```

This variable length pre-processing step, further summarised in Figure 3.3 prepares the data for the training and development of our hybrid classifier, discussed in Subsection

3.1.3. Before the discussion surrounding our classifier is presented, it is important to discuss our definition of the playstyle in-game classification problem, presented in Subsection 3.1.1.

3.1.3 Playstyle Classification

In this section we present our hybrid probabilistic supervised learning approach, which uses Bayesian Inference informed by a K-Nearest Neighbours (KNN) based likelihood, that is able to classify players in real time at every step. At any given point in a game, there exists a belief over which playstyle the player is currently using. In other words, there is a probability distribution across the playstyles which are known to exist in the game. At the start of the game episode the probability belief distribution across the playstyles will be equally likely. As the player takes actions within the game the playstyle belief probabilities will shift in the direction of which ever playstyle is likely to have taken the observed actions. For this reason, Bayesian Inference is a suitable technique for modeling and tracking these changes in playstyles.

Bayesian Inference is an inference technique which uses Bayes' theorem to track the probability of a hypothesis. The probability of this hypothesis is updated as new evidence is observed, whilst taking into account the historically observed evidence [Bolstad and Curran 2016]. Equation 3.2 illustrates how Bayes' theorem is applied to the playstyle identification problem:

$$P(playstyle|observation) = \frac{P(observation|playstyle)P(playstyle)}{P(observation)} \tag{3.2}$$

Where $playstyle \in \Gamma$, $observation \in \Psi$, and where $P(observation|playstyle)$ is the likelihood, provided by the KNN classifier (where our KNN model is fit using the labeled play log data in Υ). The likelihood probability can be seen as how consistent the new observation is with the existing historic observations made by players with this playstyle. The prior probability, $P(playstyle)$, is the initial or best guess estimate of the probability of the hypothesis before any observation or new evidence is observed. The posterior probability, $P(playstyle|observation)$, is what we aim to track and update. The posterior aims to indicate which playstyle the player belongs to, given the current observation. The marginal likelihood, $P(observation)$, is the probability of the observation without taking the playstyle under consideration into account. Since the marginal likelihood is not dependent on the playstyle under consideration, it is the same value for the probability belief distribution across all the playstyle classes. In our case, we calculate marginal likelihood using the total law of probability and sum the results from the prior and likelihood probability calculations across the playstyle classes [Bolstad and Curran 2016]. The playstyle class with the highest probability value in the posterior probability distribution is what our hybrid classifier outputs as the predicted playstyle for the observed player action.

A core component of Bayesian Inference is the likelihood probability function, which is provided by our KNN classifier, fit using the labelled play log data from steps A-D in Figure 3.1. When given an unseen play log entry, our KNN classifier returns the

playstyle classification as well as the probability distribution of this classification across the playstyle label classes. This output probability distribution is used as the likelihood probability distribution.

The predicted playstyle class label is determined by looking at the K nearest neighbours surrounding an observed play log entry. This measure of nearness or closeness between the observed play log and its neighbours is called the distance metric. A distance metric such as the Euclidean distance is used to calculate the distance, or similarity, between the observed play log vector and the surrounding K play log vectors. The most common playstyle label assigned to these surrounding K neighbour play log vectors is output as the observed play log's playstyle [Mitchell 1997]. The number of neighbours, K, is chosen before the model is fit on the labelled play log data. The number of neighbours to use for K can be determined empirically or by using techniques such as the Elbow method or Silhouette coefficient [Kodinariya and Makwana 2013].

Our KNN model returns a playstyle class label for the observed play log, along with the probability distribution across the playstyle classes for the observed play log - which is used as our likelihood. Each probability estimate in this KNN predicted probability distribution indicates how likely the observation belongs to each respective playstyle class. This KNN probability distribution across the playstyle classes is calculated by finding the K nearest neighbours surrounding the observed play log. Then the probability estimate of each class label is calculated by dividing the number of occurrences of each class by the number of neighbours [Bishop 2006]. In other words, for a playstyle class ($playstyle$), the play log observation ($observation$), and the K surrounding neighbours of the observation, the probability estimate, P_e, of the playstyle class is given by:

$$P_e(playstyle) = \frac{Number\ of\ neighbours\ surrounding\ observation\ with\ label\ playstyle}{K} \tag{3.3}$$

Let us assume we have:

- a fitted KNN model with $K = 5$
- the following playstyle classes: *Brave treasure hunter* and *Monster killer*
- the following play log observation: $[0, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10]$

Where each index in this play log observation corresponds to a metric within a given game (this can be a score, player health, or numbers of items collected). In order to calculate the probability distribution across the playstyle classes for the observed play log vector, we consider the five nearest neighbours to our observation since $K = 5$. Suppose the five nearest neighbours are:

- Neighbour 1 which belongs to the playstyle class *Brave treasure hunter*
- Neighbour 2 which belongs to the playstyle class *Brave treasure hunter*
- Neighbour 3 which belongs to the playstyle class *Brave treasure hunter*

- Neighbour 4 which belongs to the playstyle class *Monster killer*
- Neighbour 5 which belongs to the playstyle class *Monster killer*

Next the estimated probabilities for each playstyle class are calculated using Equation 3.3.

1. Three of the five nearest neighbours belong to *Brave treasure hunter*
   Therefore, $P_e(\textit{Brave treasure hunter}) = \frac{3}{5} = 0.6(60\%)$
2. Two of the five nearest neighbours belong to *Monster killer*
   Therefore, $P_e(\textit{Monster killer}) = \frac{2}{5} = 0.4(40\%)$

The underlying probability distribution across the playstyle classes, for the given observation, is thus: 0.6 for *Brave treasure hunter* and 0.4 for *Monster killer*. This predicted distribution means that the play log observation has a sixty percent chance that the play log observation belongs to the *Brave treasure hunter* playstyle class and a forty percent chance that it belongs to the *Monster killer* playstyle class. Since the *Brave treasure hunter* class had the highest probability value in the distribution, it is returned as the final playstyle class prediction. This predicted playstyle probability distribution from the KNN classification is used as the Bayesian likelihood probability during the posterior belief update. It is important to note that the implementation of the KNN likelihood function should not return a zero probability estimate for any of the classes. If probability estimate value of zero is returned for a playstyle class, this will result in no futher updates in the posterior belief update for this class; since the prior will be multiplied by a zero likelihood, as defined in Equation 3.2. A default minimum probability estimate of one percent should at least be returned.

Suppose we have the following probability distribution returned, where none of the K nearest neighbours had the *Heal and Run*, *Monster killer*, *Safe runner*, and *Wreckless runner* playstyle class labels:

- *Brave treasure hunter*: 0.66667
- *Heal and Run*: 0.0
- *Monster killer*: 0.0
- *Pure treasure hunter*: 0.33333
- *Safe runner*: 0.0
- *Wreckless runner*: 0.0

Our KNN model ensures that a zero value probability estimate is not returned by assigning a minimum probability estimate to the zero value probability estimates and then adjusting the distribution accordingly through equal subtraction. This is achieved by assigning a default probability estimate value, in our case we use the default value of $0.01(1\%)$, to each of the zero value probability estimates. Next the *total number of*

zero value probability estimates should be determined. In our case, we have four probability estimates with a zero value. We thus need to allocate $0.04(4\%)$ of the probability distribution to the zero value probability estimates. We refer to this value as the *total allocated probability*. This *total allocated probability* value of $0.04(4\%)$ was calculated by multiplying the default probability estimate value of $0.01(1\%)$ by the *total number of zero value probability estimates*, which is 4. To update the distribution accordingly through equal subtraction we calculate the amount to subtract by dividing the *total allocated probability* by the *total number of non-zero value probability estimates*; which in our case is $0.04/2 = 0.02$. This value, of 0.02, is then subtracted from each non-zero value probability estimate. We thus obtain the updated probability distribution of:

- *Brave treasure hunter*: 0.64667
- *Heal and Run*: 0.01
- *Monster killer*: 0.01
- *Pure treasure hunter*: 0.31333
- *Safe runner*: 0.01
- *Wreckless runner*: 0.01

For an observed player action, we require the ability to reason about which playstyle this observation belongs to. Furthermore, since we aim to enable real time in-game classification, we need the respective likelihood probability function to be responsive so that it can rapidly shift the probability belief distribution towards the most likely playstyle class. For this reason KNN is a suitable choice since the KNN classification process is light-weight, explainable, and the KNN classification is only dependent on the K nearest neighbours surrounding the input observation [Mitchell 1997]. In this way, the playstyle belief is purely dependent on similar neighbours (play log entries in our case) which allows for the KNN prediction to change rapidly and independently of any previously made predictions. The intuition is that similar playstyles should produce similar play log data. In order to fit the KNN model, we make use of the labeled play log data produced from the Player Action Data Processing step, outlined in Subsection 3.1.2.

The pre-processing step performed ensures that, in the training set, each play log entry which represents a given state has the corresponding playstyle label which produced the entry. Since the training data is single state play log entries which conform to the Markov property, the KNN model fit on this training data is thus able to classify single state play log entries. This capability enables our approach to classify play log entries without the need for historic past state information. This is desirable since we want our companion agent, which will make use of our hybrid playstyle classifer, to be able to assist from the start of the game and without the need to observe several actions first.

In Figure 3.4, which illustrates our Hybrid Supervised Bayesian Inference process, the prior distribution is initialised to equally likely at the start of the Bayesian belief update

process. When a new player action is observed, our KNN model returns the likelihood probability belief distribution across the playstyle set. Furthermore, the marginal likelihood is calculated using the law of total probability. Using Bayes' theorem the posterior probability is obtained. The highest posterior playstyle belief probability is returned as the playstyle prediction by our hybrid classifier. When a new observation is obtained, the prior becomes the posterior and the new posterior probability distribution is calculated.

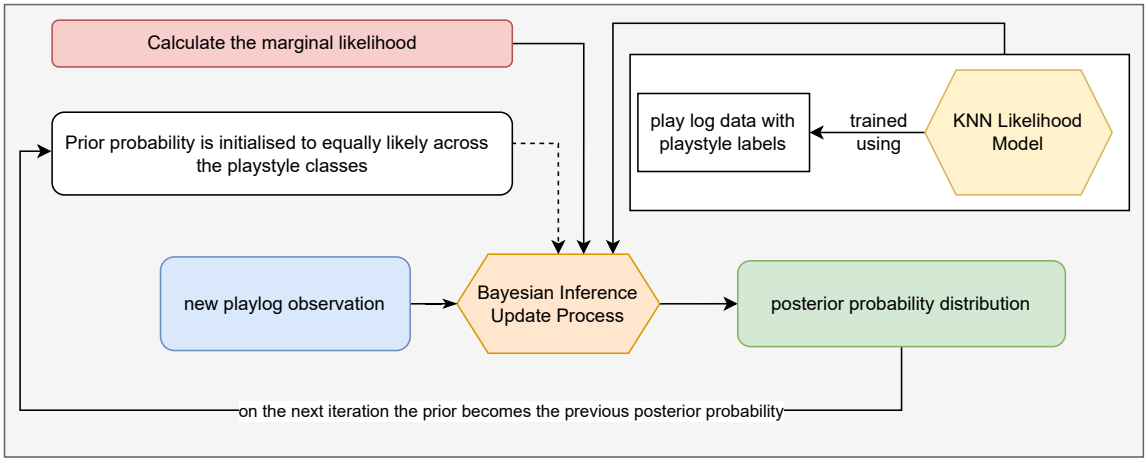


Figure 3.4: Bayesian Inference posterior belief update process with our KNN based likelihood probability model

In Section 3.2 we present our method for solving the 'Learning to Assist' problem which makes use of our hybrid classifier to inform the companion agent action selection such that a suitable action is returned for the observed playstyle.

3.2 Learning to Assist Method

A successful companion agent requires the ability to complement their human collaborator's playstyle. Depending on the human collaborator's playstyle this companion agent's playstyle may complement and aid the player in their playstyle specific goal, or it may not aid much at all, or the agent's playstyle may hinder the player from obtaining their goal. Which suggests that for a given game which has several ways in which the game can be played, some playstyles are complementary and some may not be. The intuition is that a mapping function can be learned which maps a given playstyle to a complementary playstyle. Once this mapping is created, a suitably mapped companion agent policy can be used to respond accordingly to the observed playstyle - classified using our hybrid playstyle identification classifier (outlined in Section 3.1). Our method thus consists of three steps:

- Companion Agent Policy Creation in Subsection 3.2.2
- Human Proxy Playstyle and Companion Agent Policy Mapping Process in Subsection 3.2.3

- Companion Agent Action Selection in Subsection 3.2.4

The core problem we aim to solve is the problem of determining the best, or most complementary, action response for a given playstyle. We define this problem as the 'Learning to Assist' problem: where given a set of agent policies which can play a given game, determine the best policy which best complements the observed playstyle. We formally describe this problem in Subsection 3.2.1.

3.2.1 Learning to Assist Problem Definition

This section describes the terminology and definitions necessary for defining the 'Learning to Assist' problem. Given a set of companion agent policies which can play a given game, we aim to determine the best policy which best complements the observed playstyle. In other words, we aim to find a function which maps a player's playstyle to a companion agent policy. In order to find this mapping between a set of playstyles and the set of companion agent policies, we define two sets. We make use of the Playstyle Set Definitions for Γ and P_g from Subsection 3.1.1 and define a new set containing companion agent policies using P_g:

- We use the set $\Gamma \subseteq P_g$ which represents a set of playstyles which capture human player behaviours within a given game.
- We define the set of companion agent policies as $\Pi \subseteq P_g$

Where, P_g is the global set of all possible playstyles for a given game g. It is important to note that we are mainly concerned with determining the best response, or mapping, for a given playstyle in set Γ from the set of policies in Π. Additionally, our method is not concerned with optimal policy creation for the policies in Γ and Π. For these reasons, we are only concerned that there exists these two sets Γ and Π. Furthermore, the sets Γ and Π may be disjoint, or they may be overlapping sets. As part of our experiment, in Section 4.2, we simulate the human playstyle behaviours in Γ using rule-based agents and for the policies in Π we make use of Deep Q-learning agents to represent the companion agent policies.

For each playstyle in the fixed and defined playstyle set Γ we aim to find a suitable companion agent policy in Π. A playstyle and companion agent policy pair is complementary if and only if the joint expected reward R is greater than or equal to every other playstyle-policy combination's expected reward. Let $p^* \in \Gamma$ and $\pi^* \in \Pi$. The pair (p^*, π^*) is complementary if and only if:

$$\forall\, p \in \Gamma\,, \forall\, \pi \in \Pi : R(p^*, \pi^*) \geq R(p, \pi) \tag{3.4}$$

In order to inform the joint expected reward, R, we define a set ϖ which contains the respective reward functions which represent the internal motivation behind each playstyle in Γ. In other words, for each $p \in \Gamma$ define a reward function $r \in \varpi$ which represents the internal motivation for p within a given game g.

Using this definition of a complementary playstyle-policy pair, we create a function

Λ, which for every input playstyle in Γ, returns the complementary agent policy from Π:

$$\Lambda : \Gamma \rightarrow \Pi\,, \tag{3.5}$$

such that $(p, \Lambda(p))$ is complementary $\forall p \in \Gamma$. From a training and evaluation perspective a portion of the levels for game g are used to train the policies and inform the human proxy playstyle and companion agent policy mapping process. Furthermore, a portion of the levels are used to evaluate the created companion agent's ability to adapt and assist the various playstyles in Γ. We respectively refer to these levels as L_{train} and L_{test}.

3.2.2 Agent Policy Set Creation

In order for our companion agent to be able to assist different kinds of playstyles we need a set of policies from which to choose. The Agent Policy Creation step in our method is focused on creating the policies and agent logic within the respective Π and Γ sets, as defined in Subsection 3.2.1. The companion agent policy set is represented by Π and the set Γ contains the playstyles representing the human playstyles within a given game. Furthermore, each playstyle defined in Γ should have a reward function created which best encapsulates what the playstyle values and finds rewarding within the given domain. The intuition is that each reward function, in the set ϖ, represents the hidden internal reward which drives the given playstyle. Furthermore, the defined playstyle reward functions are used when calculating the joint expected reward R for each playstyle-policy pairing, performed in Subsection 3.2.3. The respective agent logic and policies within the Γ and Π sets are developed and trained using the training level set, L_{train}.

3.2.3 Human Proxy Playstyle and Companion Agent Policy Mapping Process

In order to create the Playstyle-Policy Mapping, Λ, we need to evaluate each Playstyle-Policy pair and record the respective reward value outcomes. We make use of the L_{train} level set for the game g to create the initial mapping. It is important to note that the mapping is performed offline.

An assumption made by our method is that for each playstyle a reward function exists which encapsulates what the playstyle finds rewarding. Creating a reward function which represents a given human behaviour is challenging in itself. For this reason, we make use of playstyle agents to represent human players. As discussed, in Chapter 2, a diverse set of playstyle proxy agents can successfully be created and represent human players in a given game. Furthermore, in the situation where no historic player data exists, playstyle proxy agents with defined reward functions are sufficient. As play log data becomes available inverse reinforcement learning techniques can be explored to learn a given playstyle reward function [Arora and Doshi 2021]. This is out of scope and will be addressed as part of future work.

Each playstyle-policy pair plays each level in the training level. After each episode in the respective training level, the total joint-reward obtained by the playstyle-policy pair is recorded. The playstyle-policy pairing with the highest average joint-reward is selected as the Playstyle-Policy mapping in Λ. Algorithm 5, in Appendix B, illustrates this human playstyle and companion agent policy mapping process.

3.2.4 Companion Agent Action Selection

The companion agent action selection is dependent on the existence of a model Ω_Υ which can at every $step$, within a level l, in the game g classify the unseen play log entries in Ψ, of agent α, to a playstyle in Γ, as defined in Equation 3.1 in Subsection 3.1.1. This is the outcome from Research Objective I. Furthermore, the companion agent action selection is dependent on the creation of the Playstyle-Policy Mapping, Λ, defined in Subsection 3.2.3. For a given step i within an episode, the companion agent makes use of the Ω_Υ playstyle classifier to classify the observed player. The Playstyle-Policy Mapping, Λ, is used to find the mapped policy response to this classified playstyle. The state at i along with the mapped policy is used to determine the respective action. This process is illustrated in Algorithm 4.

Algorithm 4 Companion Agent Action Selection Process

Require: Λ, Π, Ω_Υ, observed_playlog $\in \Psi$, $state_i$

```
1: companion_agent ← AdaptiveCcompanionAgent.initialise()
2: companion_agent.load_policies(Π)
3: predicted_playstyle ← companion_agent.classify_player(Ω_Υ, observed_playlog)
4: companion_agent.take_action(state_i, Λ, predicted_playstyle)
```

From a technical perspective, the adaptive companion agent class contains the respective agent policies and the hybrid classifier (Lines 1-2 in Algorithm 4). A call to the hybrid classifier is then made to classify the observed player action play log to a playstyle class (Line 3 in Algorithm 4). At runtime the adaptive companion agent is passed the latest environment observation containing the current state and the previous player action which brought about this state. The output playstyle prediction is then passed into the playstyle to policy mapping function, which returns the companion agent policy to use. The adaptive agent returns an action using the current state and the mapped policy for the predicted playstyle (Line 4 in Algorithm 4).

As part of our experiment, presented in Section 4.2, we develop two playstyle adaptive agents, a mimic based companion agent which mimics the identified playstyle and our fully adaptive companion agent which responds to the identified playstyle using the Playstyle-Policy Mapping. The static companion agent policies contained in Π are used as a baseline against our mimic and fully adaptive companion agents, using the L_{test} level set from the game g (which is Collaborative Minidungeons, as presented in Subsection 4.2.1).

Chapter 4

Experimentation and Results

4.1 Empirical Evaluation of Playstyle Classification

The ability to correctly classify or identify the player style is a necessary requirement for our adaptive companion agent. In order for a playstyle-aware adaptive agent to confidently assist the player in real time with a given task, the companion agent firstly requires a mechanism for correctly classifying or identifying the player style, which is our our Research Objective I. Our playstyle identification method outlined in Chapter 3, Section 3.1, presents our method for solving the problem of playstyle classification.

In order to fully evaluate our playstyle identification framework two experiments are presented. Each experiment tests the ability of our framework to account for the two situations related to the availability of existing player play log data. Since playstyles in games are identified based on the observed in-game event information our approach makes use of a *play log*, which contains the essential game metrics required to identify or discriminate between playstyles [Iawasaki and Hasebe 2021][Iwasaki and Hasebe 2022]. Our playstyle identification framework is able to account for the following two cases: a general case where no play log data exists, and the special case where already existing player action play log data exists.

The general case commonly occurs since the availability of player action data is usually not available during the early stages of game development. Additionally, testing with human players can be a costly exercise. Hence, for the general case we need to generate synthetic player action trajectory data using human behavioural or human proxy agents. The special case occurs when actual human player action data exists. The special case is less common since human player action data is hard to obtain or is often not made public by game publishers. Therefore, for the special case where there is no existing play log data, traditional unsupervised clustering is used to obtain a playstyle label for the respective play log data. Before the general case experiment we discuss the MiniDungeons domain in Subsection 4.1.1. Subsection 4.1.2 presents the general case experiment setup, where we make use of the rogue-like dungeon exploration game called MiniDungeons to test and apply our playstyle method. In Subsection 4.1.3 we discuss the MiniDungeons experimental results, our general case evaluation. For the special case evaluation we make use of a natural dataset containing human player ac-

tion data[1] from the video game Super Mario Bros. The real human player data allows for us to evaluate the scalability of our framework to handling real player data within a more complex game domain. In Subsection 4.1.4 we present the special case human player experiment setup and implementation. In Subsection 4.1.5 we discuss our, special case, Super Mario Bros experimental results.

As part of our experiments we develop several variants of our hybrid supervised Bayesian classifier based on different play log instantiations which we have defined. Furthermore from an evaluation perspective we compare our approach to the comparative unsupervised approach by Ingram *et al.* [2022].

4.1.1 The MiniDungeons Game Domain

MiniDungeons is a turn-based top-down tile based dungeon exploration game, created as a benchmark research domain for modeling and understanding human playstyles [Holmgård *et al.* 2021][Holmgård *et al.* 2014a]. Each dungeon level contains monsters which can be battled, treasure which can be collected, and potion which can be used to restore health, as shown in Figures 4.1 and 4.2.

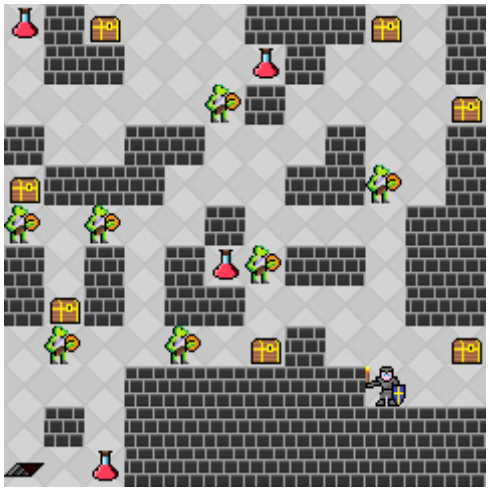


Figure 4.1: MiniDungeons (Level 9)

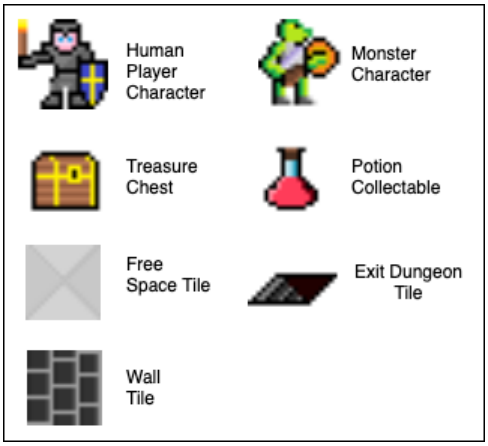


Figure 4.2: MiniDungeons Legend

Player interaction with the monster, treasure, or potion occurs immediately when the player occupies the same item tile. The monster, treasure, and potions, and other world elements are stationary. Monsters randomly deal between five and fourteen points of damage, while potions grant a fixed amount of ten health points, whilst the player health is not full. Players have a maximum of forty health points. When treasure is collected the in-game treasure counter is incremented. When the players health reaches zero, the game is over. A dungeon can be exited by reaching the exit tile. In

[1] http://guzdial.com/datasets.html

total there are ten levels and one tutorial level [Holmgård *et al.* 2014c][Holmgård *et al.* 2014b]. Figure A.1, in Appendix A, illustrates the layouts for the respective eleven levels. We use a python, OpenAI Gym compatible, re-implementation[2] [Iwasaki and Hasebe 2022]. The dynamics of the gym-md environment can be seen in Figure 4.3.

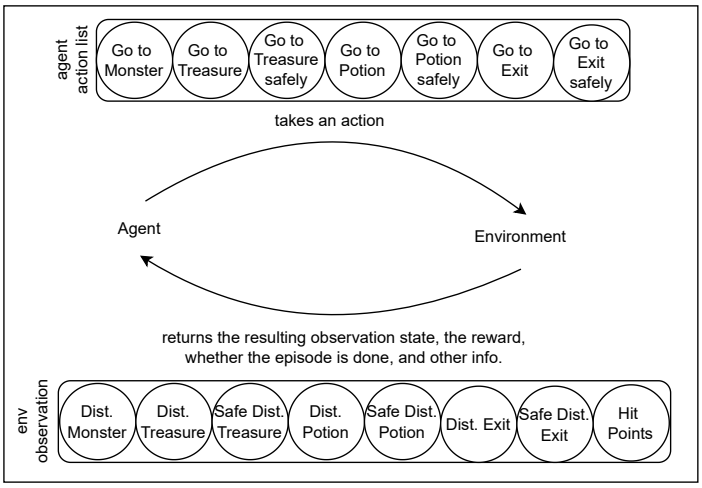


Figure 4.3: MiniDungeons Gym Environment Dynamics

As illustrated in Figure 4.3 the agent, or player, has an action list with the respective input actions. The agent thus chooses an action which is then passed to the MiniDungeons game environment. The game environment then applies the agent action accordingly.

An action within the MiniDungeons environment is represented as a list containing seven floating point values which correspond to the following actions: go to the monster, go to the treasure, go to the treasure (avoiding monsters), go to the potion, go to the potion (avoiding monsters), go to the exit, and go to the exit (avoiding monsters). The highest action value in the action list is first attempted, if this action cannot be taken in the given state then the next highest action is selected and so on. These actions are encoded by the MiniDungeons gym environment. Each action goes to the respective nearest target (monster, treasure, potion, or exit). The path taken towards the given target can either be safe or unsafe. A safe path does not contain monsters, whereas an unsafe path does.

Besides applying the agent action, the environment also returns: the resultant observation or state, the reward received for the input action, a boolean indicating whether the level episode is complete or not, and other info such as the latest play log. The resultant environment observation or state is represented as a list containing the player hit points (HP) and distances from the agent to the nearest world elements (distance to nearest monster, and the safe/unsafe distances to the nearest treasure, potion, and

[2]https://github.com/ganyariya/gym-md

exit).

During our work we have contributed[3] documentation and several bug-fixes and enhancements to the open source gym-md project, and as a result have been made a project maintainer by the owner. For more information regarding the MiniDungeons gym environment please see the repository's readme[4] and the respective function documentation for more detail. In Subsection 4.2.1 we discuss the extensions we implemented to make MiniDungeons a collaborative game in order to evaluate the companion agent.

4.1.2 General Case: MiniDungeons Experiment

From a play log instantiation perspective we define two types of play logs, each of which measure the player interaction at different granularities:

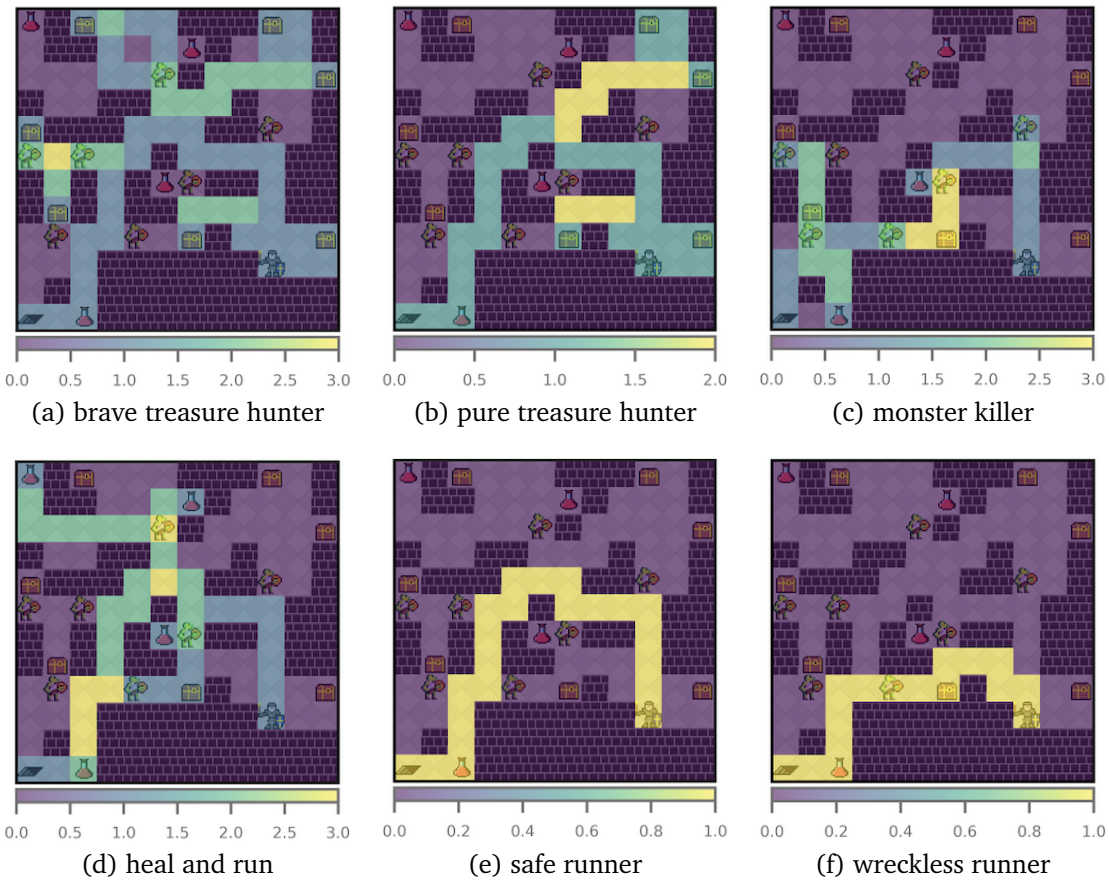
- Low level visitation grid (LLVG): is a two-dimensional vector which tracks the number of visits made to each position in a level (i.e. models at the player action level [Bakkes *et al.* 2012]). For a given MiniDungeons level the LLVG play log dimensions will be equal to the level's row and column count.
- Tactic level information (TLI): is a one-dimensional vector which stores the number of times a specific action type is taken, as well as the number of times the player visits each level cell or square type [Iawasaki and Hasebe 2021]. The TLI play log aims to track the higher level player tactics [Bakkes *et al.* 2012].

In order to generate these two respective play log datasets we define a set of six player proxy agents. The created playstyles are based on the main objectives available in a given level of MiniDungeons, for example: collect treasure, restore hit points (HP) by collecting potions, battling monsters for experience points, and reaching the dungeon exit. Furthermore, the choice of our playstyle proxy agents is informed by the clustering analysis work done by Iwasaki and Hasebe [2022] who evolve multiple human proxy agents without predefining the desired playstyle or reward function. The core playstyles generated by their approach are: a *runner* based playstyle, which prioritises exiting the dungeon as quick as possible, a *completionist* player, which aims to interact with treasure, monster, and potion elements within the dungeon, a *treasure centric* based playstyle, which focuses on collecting treasure, and a *safty-first* playstyle, which focuses on collecting treasure and potions.

In addition we have treasure centric agents called *brave treasure hunter* and *pure treasure hunter*. The *pure treasure hunter* only collects unguarded treasure, whilst the *brave treasure hunter* will collect treasure guarded by monsters and replenish hit points if needed. The *monster killer* playstyle agent is concerned with battling as many monsters as possible without dying. If the monster killer agent's hit points is low, potions will

[3] https://github.com/ganyariya/gym-md/graphs/contributors
[4] https://github.com/ganyariya/gym-md

be collected to restore health. The *heal and run* agent aims to collect all the potions in a given level. The *safe runner* and *wreckless runner* playstyles respectively aim to exit the dungeon as fast as possible. The *safe runner* agent takes the shortest available safe path to the dungeon exit (i.e. a path which does not encounter any monsters), whilst the *wreckless runner* agent will take the shortest path to the dungeon exit. Figure 4.4 illustrates the respective playstyle movement trajectories within the nineth level of MiniDungeons.



(a) brave treasure hunter (b) pure treasure hunter (c) monster killer

(d) heal and run (e) safe runner (f) wreckless runner

Figure 4.4: Playstyle movement trajectory heatmap for MiniDungeons (Level 9).

We aim to evaluate the playstyle classifiers in two ways. As a rationale for the two types of evaluations suppose we have the following situation: a game developer would like to adapt non-player character attack patterns, during the encounter with the player, within the game's existing levels. In this situation, which we refer to as the seen case, the game developer can train and deploy the classifiers and adapt the game accordingly. This seen case is an appealing evaluation in situations where a game's levels are known and defined. Next let us say that the game has been released successfully and our classifiers have been deployed and shipped with the game's initial release. In this situation our second evaluation, which we call the unseen case, may assist the game developer. As new levels are added to the game via updates, the classifiers which can generalise to these new *unseen* levels do not need to be re-trained or re-deployed. The unseen case caters for when new levels have been created after the classifier has been developed.

We therefore allocate the MiniDungeons levels[5] in the following way:

- L_{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)
- $L_{seentest}$ (md-strand_2-v0, md-holmgard_0-v0, md-holmgard_3-v0, md-holmgard_5-v0, md-holmgard_7-v0, md-holmgard_8-v0)
- $L_{unseentest}$: (md-holmgard_1-v0, md-holmgard_2-v0, md-holmgard_4-v0, md-holmgard_6-v0, md-holmgard_9-v0, md-holmgard_10-v0)

The levels assigned to the L_{train} set were selected in order to reserve the Holmgard levels for the seen and unseen evaluations. Each level run was repeated ten times in each respective evaluation case.

We develop two variants of our Hybrid Bayesian Supervised Learning classifier based on the LLVG and TLI play log instantiations, called Hybrid (LLVG) and Hybrid (TLI). In our hybrid classifier we make use of a KNN model to output a probability belief distribution across the playstyle classes - which acts as the hybrid classifier's Bayesian likelihood function. Our Hybrid (LLVG) likelihood probability function is informed by a KNN model fit using the LLVG play log representation, which we refer to as KNN (LLVG). Similarly our Hybrid (TLI) classifier's likelihood probability function is informed by a KNN model fit using the TLI play log representation, which we refer to as KNN (TLI). In other words, the KNN (LLVG) model classifies the observed play log entry using the nearest or most similar log entries in the form of the LLVG play log. Similarly, the KNN (TLI) model classifies using the nearest entries in the form of the TLI play log. For both the KNN (LLVG) and KNN (TLI) classifiers the number of nearest neighbours used is three (i.e. k=*3*, determined empirically during experiment).

In order to explore the capabilities of our hybrid classifier we perform an ablation study where we compare our hybrid classifier to the respective KNN and Bayesian Inference parts which make up our hybrid classifier. In other words, since our hybrid classifier makes use of Bayesian Inference informed by a KNN based likelihood function, we would like to compare our hybrid classifier to a stand-alone KNN classifier and to a standard Bayesian Inference based classifier which uses a traditional likelihood function informed by historic observations.

For the stand-alone KNN classifier comparison we use the KNN (LLVG) and KNN (TLI) models created for our hybrid classifier. The stand-alone KNN classifier allows for us to see the strengths and weaknesses of our hybrid classifier when the Bayesian Inference process is not present. Similarly for the stand-alone Bayesian Inference based classifiers, we swap the KNN based likelihood for a traditional likelihood function informed by historic observations. Two kinds of the stand-alone Bayesian Inference based classifiers are developed based on the LLVG and TLI play log instantiations: respectively

[5] https://github.com/ganyariya/gym-md/blob/main/README/resources/md_stages_screenshots/README.md

called Bayes (LLVG), and Bayes (TLI). By removing the KNN based likelihood function we can assess the the strengths and weaknesses of the Bayesian Inference process when the KNN likelihood is not present.

For the Bayesian Inference based classifiers, Bayes (LLVG) and Bayes (TLI), the initial prior probability belief distribution is equally likely across the playstyles. The likelihood probability that the in-game player observation belongs to a specific playstyle is informed by the historic LLVG and TLI play logs respectively. The posterior probability is then updated using the likelihood probability and the prior probability distribution. This process of updating the playstyle probability belief distribution is repeated as new evidence (i.e. the player observation) is observed.

In our initial evaluations, the stand-alone KNN classifiers outperformed the Bayes classifiers, as shown in Figure 4.5. Upon careful inspection we attribute the poorer Bayesian Inference performance to the likelihood probability function not having enough historic data to discern and provide probabilities which can adequately influence the playstyle belief distribution. The problem with the Bayes classifier is that in unseen situations the default equally likely probability was returned by the likelihood function. Since the returned likelihood probability distribution is equally likely, during the posterior belief update, it has little effect on the resultant posterior probability belief distribution. This resultant posterior probability belief distribution would effectively remain the same as the previously calculated posterior - thus resulting in no change in the playstyle classification.

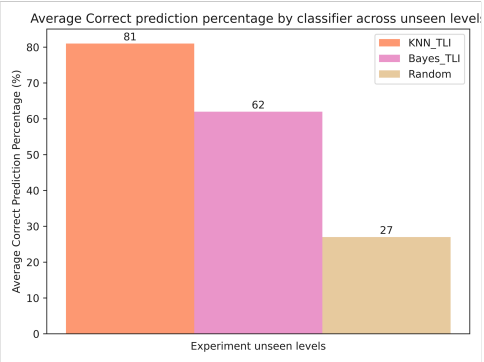


Figure 4.5: Average correct prediction percentage for the KNN (TLI) classifier and Bayes (TLI) classifier before the cosine similarity lookup, across an unseen level experiment run.

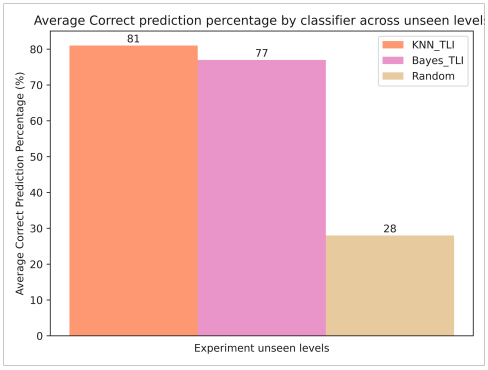


Figure 4.6: Average correct prediction percentage for the KNN (TLI) classifier and Bayes (TLI) classifier with the cosine similarity lookup, across an unseen level experiment run.

We improve the likelihood probability function of the Bayes classifier by including a cosine similarity lookup for nearest similar observations, as shown in Figure 4.6. We observe an average improvement of approximately fifteen percent between the Bayes (TLI) before and after the inclusion of the cosine similarity lookup. In subsequent experiments we make use of this improved Bayes (TLI) with the cosine similarity lookup.

The cosine similarity lookup works as follows: when a new player action is observed, if the observation has not been seen before, a search is applied across the historic data. Then the historic play log observation with the closest cosine similarity is used to determine the likelihood probability of the observation, instead of just returning the default equally likely probability distribution. This cosine similarity search results in a more informed probability likelihood. The likelihood probability function used to inform the posterior playstyle belief update is a core component within the Bayesian Inference process.

In order to further benchmark our hybrid approach, two baselines are used. The first baseline is a simple 'random' classifier which will at each step make a random prediction on the player's playstyle. The second comparative baseline is an unsupervised approach which makes use of an LSTM-autoencoder to cluster the player action trajectories [Ingram *et al.* 2022] The Autoencoder is used to project variable length trajectories in a uniform latent space which is then used for the clustering step. Although, this approach is unsupervised, we used the ground truth labels generated for both the Mario and the MiniDungeons domain in order to compute the performance of this model. We train a classifier by Ingram *et al.* [2022] based off the TLI play log representation which refer to as the Ingram (TLI) classifier.

In summary we use the following classifiers in our general case experiment:

- Hybrid classifier: Hybrid (LLVG) and Hybrid (TLI)
- KNN classifier: KNN (LLVG) and KNN (TLI)
- Bayesian Inference classifier: Bayes (LLVG) and Bayes (TLI)
- Ingram (TLI) classifier
- Random classifier

The classifiers are all trained on the L_{train} levels and evaluated on the $L_{seentest}$ and $L_{unseentest}$ levels. Each level run was repeated ten times for each playstyle within each respective evaluation case.

4.1.3 General Case: MiniDungeons Experimental Results

As outlined in the experiment setup, we have run two kinds of evaluations on the respective classifiers. The first evaluation is run on levels which have been seen before by the respective classifiers. The seen case is an appealing evaluation in situations where a game's levels are known and defined. The unseen case caters for when new levels have been created after the classifier has been developed. The representation of the play log is key to enabling this generalisability across unseen levels.

The LLVG play log representation is at a lower granularity, than the TLI representation. The LLVG play log is level specific since the play log's dimensions and the positional relevance of the player is coupled to the specific level under consideration. This means

that, in this case, the LLVG classifier cannot be applied to unseen levels. This is why the respective LLVG classifier results are absent in the unseen evaluations (as shown in Figure 4.8 and Table 4.1). Since the TLI play log is at a higher 'tactic level' abstraction the same TLI vector can generalise across levels.

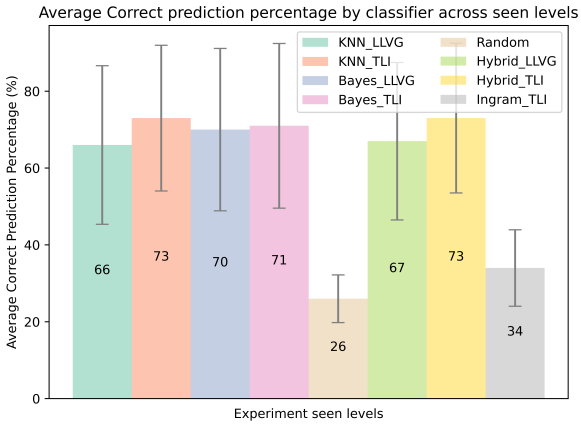


Figure 4.7: Seen Evaluation

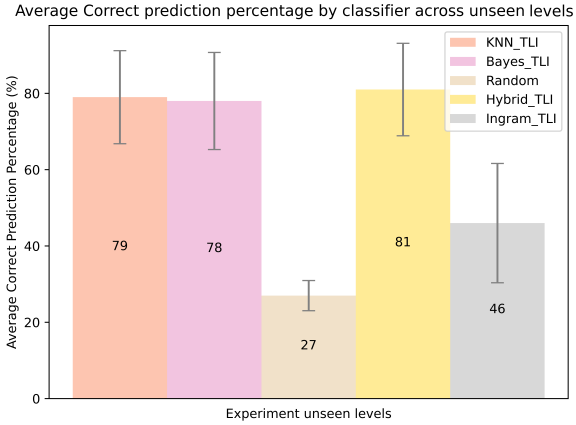


Figure 4.8: Unseen Evaluation

The results shown in Figure 4.7 confirm that in the seen case, each classifier is able to, on average, accurately predict the correct playstyle, well above the random classifier and our comparative case. The KNN (TLI) and Hybrid (TLI) classifiers marginally performs better than the other classifiers. The interesting observation is that in all three classifier types the TLI based representation resulted in better average accuracies when compared to the LLVG variants. This suggests that the TLI play log representation better captured the player interaction in comparison to the LLVG representation. Initially, we expected the LLVG representations to perform better since the play logs are better fit to each level and are at a lower granularity. However, this is not the case since the KNN LLVG performed marginally weaker.

Figure 4.8 summarises the unseen evaluation of the respective TLI based classifiers. On average our classifiers accurately predict the correct playstyle well above the random classifier and to our comparative case. The results in Figure 4.8 confirm that the higher level TLI play log representation is able to generalise to unseen levels. The Hybrid (TLI) classifier marginally outperformed the KNN (TLI) and Bayes (TLI), which we attribute to the difference in stability scores, which is summarised in Table 4.1. However, before discussing the stability score metric, it is important to note that, in both the seen and unseen evaluations, our classifiers at each step received only the latest play log entry, whilst the Ingram (TLI) classifier received the latest play log entry as well as the previous temporal play log entries. Thus highlighting the success of our framework's variable length pre-processing step in removing the dependency on previous episodic player action trajectories in order to classify the player. As highlighted in Figure 4.8 we outperform the comparative case. Although some of the poor performance associated with the approach by Ingram *et al.* [2022] can be attributed to the unsupervised nature of the model, our performance is substantially greater and can be done using only the current play log state rather than requiring the past temporal play log entries.

Table 4.1 summarises an important result relating to the average playstyle prediction stability for each classifier. At each step within a given episode the playstyle classifier makes a playstyle classification based on the observed player action. The stability of the classifier is a count of how many times, during an episode, the classifier changes its playstyle type prediction (e.g. transitioning from a Monster Killer prediction to a Brave Treasure Hunter prediction). In order to compute the stability score, during an episode, we keep track of each playstyle prediction made, by each classifier, as well as a counter. If at any point the playstyle prediction made changes from one playstyle class (or type) to another, then the stability score counter is incremented. In order to obtain the overall average stability score, we average the episodic stability score counter across the number of episodes ran per level, and then we average across the evaluation test levels.

Table 4.1: Overall Average Stability Score by Classifier

Classifier Type	**Average Stability Score**		
	Seen Levels		**Unseen Levels**
	LLVG	**TLI**	**TLI**
KNN	8.317	4.5	2.778
Bayes	**0.222**	**0.111**	**0.444**
Random	51.017		54.105
Hybrid	2.583	1.611	1.556
Ingram (TLI)	1.333		1.167

A classifier is seen to be more stable when the output playstyle prediction class change occurs less frequently. The lower the stability score, the more stable the classifier. Since the random classifier makes a random prediction at every step the stability score is high, and thus the random classifier is a less stable classifier relative to the other classifiers. Table 4.1 highlights the big difference in stability between the KNN and Bayesian approaches. We respectively attribute these to the nearest neighbour distance logic in KNN and the gradual belief update in the Bayesian update.

The Bayesian Inference based classifier at each steps tracks a probability belief distribution across the playstyle types. The playstyle which has the highest probability in the distribution is returned as the predicted playstyle. This playstyle prediction output will only change when the probability belief distribution has a new highest playstyle probability class. This posterior probability belief distribution update is why we call it a *gradual* belief update, since it may take a few more observations for a new highest playstyle probability class to emerge. In contrast, the KNN based classifier only takes into account the current observation and finds the K-nearest play log entries (neighbours) to obtain the playstyle class prediction. This means that the KNN prediction is solely dependent on the K-nearest play log entries (neighbours) and can more rapidly shift and change the playstyle prediction output.

If a more stable classifier is required, then the Bayesian approach may be better. If the rate of prediction type change is not a concern then the KNN approach can be con-

sidered. The idea behind our Hybrid classifier is that the classifier should ideally bring about good balance between the rate of change in playstyle prediction type, i.e. the stability, and the accuracy of the believed playstyle prediction. When comparing the unseen evaluation TLI stability scores of the KNN (2.778), Bayes (0.444), and Hybrid (1.556), we see that the Hybrid classifier's stability sits between the KNN and Bayes classifiers, whilst obtaining a higher average accuracy in the unseen case, as seen in Figure 4.8. This insight along with the obtained KNN and Bayes comparative accuracies and prediction stability results, suggests that a well balanced approach is to combine the two classifier types into a hybrid classifier.

The choice of classifier, in terms of stability, is dependent on how the playstyle prediction classifier is used to adapt a game accordingly. For example, when adapting a game's difficult at the end of a level or after an enemy encounter, then the prediction stability of the classifier is less of a consideration because the game adaptability occurs less frequently. However, if one is adapting the non-player character (NPC) behaviour in real time in response to player actions then the stability of the classifier is quite important. In this case of playstyle-adaptive NPC agents, players may be confused or the gameplay experience may be adversely affected if the playstyle prediction changes more often, which may result in conflicting or inconsistent NPC agent behaviour. We continue the evaluation of our framework using the Super Mario Bros human player data, in Subsections 4.1.4 and 4.1.5.

4.1.4 Special Case: Super Mario Bros Experiment

For the special case evaluation we aim to evaluate the scalability of our framework in handling real player data within a more complex game domain. We make use of a natural dataset containing human player action data[6] from the video game Super Mario Bros. We use an unsupervised clustering approach to cluster the player data. The resultant clusters can be seen as surfacing the playstyles within the given dataset, since similar play log data will be grouped together. Each cluster is assigned a label which is used as the playstyle label for the respective play log data contained within the cluster, as outlined in our method in Chapter 3. As part of this experiment, we also explore the effect the play log granularity has on the playstyle prediction and stability score.

The Super Mario Bros evaluation dataset was created as part of a user study, conducted by Guzdial and Riedl, consisting of seventy-four human players which played through twelve levels of Super Mario Bros [Guzdial and Riedl 2016]. Super Mario Bros is a video game which involves moving the main player character called Mario through a two-dimensional level traversing it from left to right whilst navigating platforms, jumping gaps, and overcoming enemies, until the end of the level is reached, as illustrated in Figure 4.9.

[6] http://guzdial.com/datasets.html

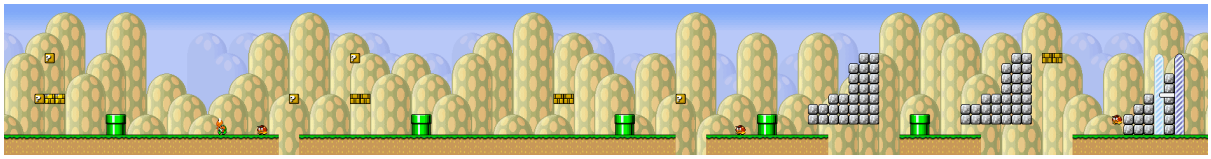


Figure 4.9: Example level from the Mario PCG dataset (Level C16)[Guzdial and Riedl 2016].

Mario is able to move left and right, jump, run, and shoot fireballs, if the enabling *fire flower* item is collected [Pedersen *et al.* 2009]. We make use of the Super Mario Bros natural human player action dataset in order to ascertain whether our framework is able to scale to a more complex domain. Furthermore, Super Mario Bros is a suitable domain for studying playstyles since there are a number of different ways to play the game. The global goal of Super Mario Bros is to reach the end of the level, however, there are coins which can be collected and various enemies which can be defeated in different ways based on collecting different items which all affect the final score obtained. There is also an exploratory element where players can try find shortcuts, hidden areas, or bonus rooms accessed by traversing green warp pipes or by breaking platform blocks to enable new pathways. Additionally there is also a level timer which also affects the player's final score. In Super Mario there are special items to collect such as the *fire flower* which enables Mario to shoot balls of fire at enemies and the *super mushroom* which gives Mario the ability to break through platform bricks and take an extra hit from enemies. Both of these items change the way game is played, for example, enemies can be vanquished in various ways, and hidden parts or shortcuts can be discovered through the breaking of platform bricks. Thus players are free to play based on what they value, be it finishing the level as fast as possible, maximising score through coin collection and enemy kills, or by exploring the level by trying to traversing green warp pipes which may lead to a hidden part of the level or bonus area.

For this evaluation we define two types of play logs, based on the TLI representation, each of which summarise the player interaction at different granularities:

- Summarised Tactic level information (S-TLI): is a one-dimensional vector which stores the number of times the player jumps, kills, runs, breaks bricks, and dies. The S-TLI is a summarised play log view of the player's in-game interactions and looks as follow:

  `['JumpStart', 'Kill', 'RunStateStart', 'BlockCoinDestroy',`

  `'Die', 'TrajectoryOccurance'].`

  These metrics are useful such that it summarises the main player interactions within the game, namely navigating platforms via jumping, killing in-game enemies, overcoming obstacles which require the player to be in a running state, collecting coins contained in blocks, tracking the number of times the player dies, and when this play log entry occurred. Furthermore, we make use of this play log since it is the representation used by Ingram *et al.* [2022] in their study. However, our intuition is that this summarised representation is too abstracted and high level to truly capture the underlying differences between the possible playstyles

present. Hence, we define a secondary play log instantiation which tracks metrics at a lower more detailed granularity, called the Extended Tactic level information (E-TLI).

- Extended Tactic level information (E-TLI): is a one-dimensional vector which is an expanded more granular form of the S-TLI tracking the occurrence of forty action and in-game interaction events, as illustrated in Figure 4.10 (A). For the purposes of this discussion and in order for us to more easily describe the E-TLI metrics, we group each E-TLI metric into a high-level logical category, as shown in Figure 4.10 (B). Each metric falls into a high level category, which we define based on the main objectives and outcomes in a typical Mario level, which are: coin collection, power-up collection which affect Mario's state, player actions taken, player deaths, level outcome and related metadata, and enemy kill related metrics. Again it is important to note that the categorisation in Figure 4.10 (B) is just for illustrative purposes and has no meaning beyond providing a logical grouping of where the respective E-TLI metrics fall in relation to gameplay and player interaction. When training the classifier models and processing the Mario dataset the complete one-dimensional vector form in Figure 4.10 (A) is used.

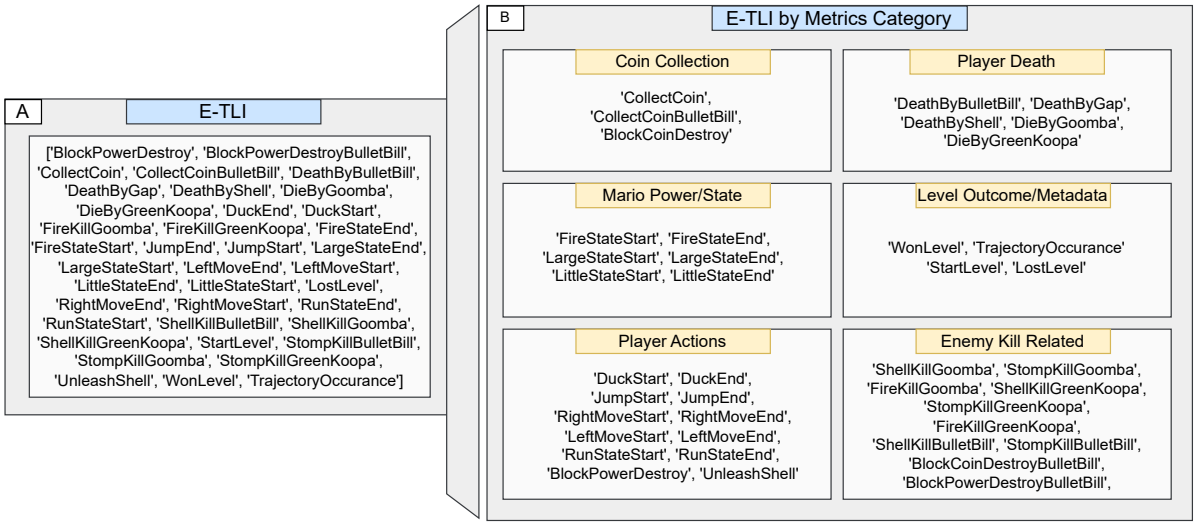


Figure 4.10: Full list of Extended Tactic level information (E-TLI) metrics shown in (A), with the same respective E-TLI metrics by category shown in (B).

To generate these two respective play log datasets[7] we processed the source Mario data and performed K-means clustering on the respective play log data to obtain the prospective playstyles within the dataset. Figure 4.11 and 4.12 respectively show the S-TLI and E-TLI PCA-Reduced Mario data after K-means clustering. This processing step was necessary as the Mario dataset is unlabeled. The efficacy of this approach to surfacing prospective playstyles, is demonstrated by applying the same unsupervised

[7]https://github.com/LJArendse/playstyle_classification_using_hybrid_probabilistic_supervised_learning

K-means clustering to our MiniDungeons dataset. Here we were able to successfully extract the six desired clusters which correspond to the six playstyle proxy agents, as shown in Figure 4.13. The respective values of k was determined using the silhouette coefficient and elbow method [Kodinariya and Makwana 2013].

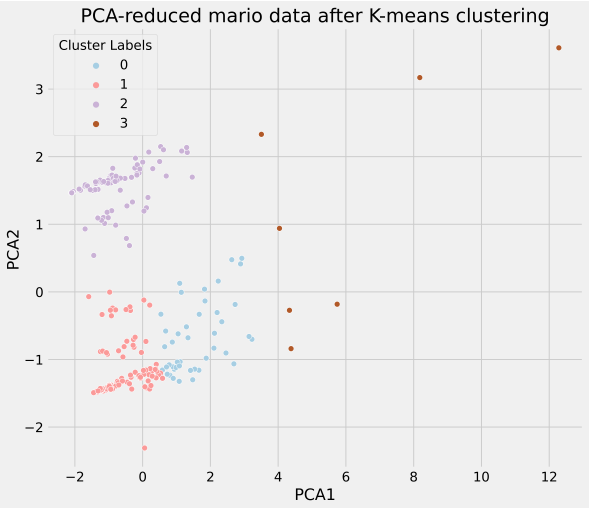


Figure 4.11: PCA-Reduced Mario (S-TLI) data after clustering.

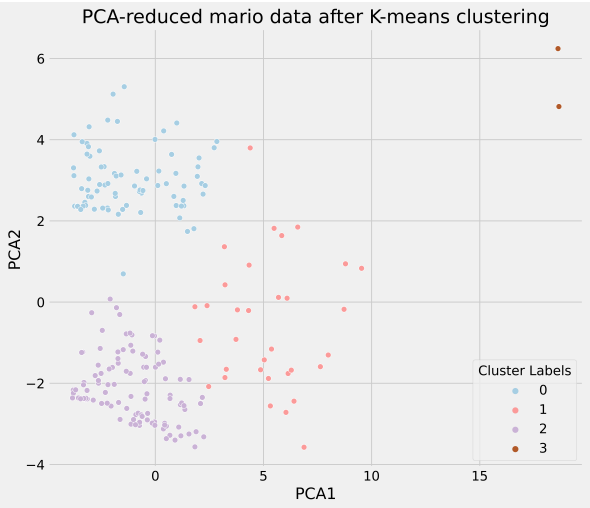


Figure 4.12: PCA-Reduced Mario (E-TLI) data after clustering.

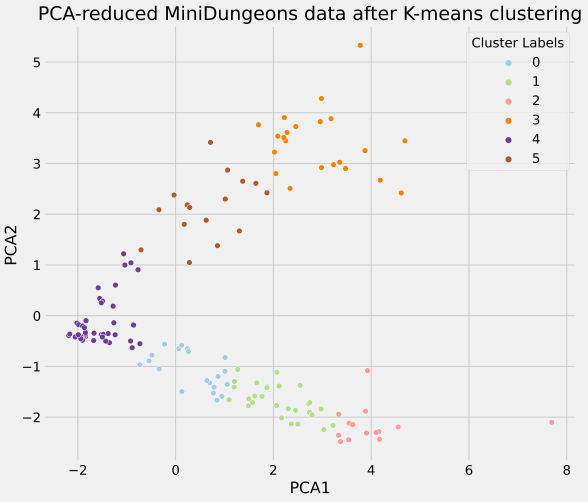


Figure 4.13: PCA-Reduced MiniDungeons (TLI) data after clustering.

It is interesting to see that the K-means clustering on the S-TLI and E-TLI play log representations both resulted in four clusters, despite being at different levels of granularity. This highlights a possible commonality or relationship in the underlying play log data. The cluster labels obtained will be used as the playstyle labels during training. The idea as shown in the MiniDungeons case, Figure 4.13, is that similar play log entries belong

to the same playstyle cluster. Thus unseen play log entries should fall within a cluster containing the most similar play log data. From a training and evaluation perspective we took the player action Mario dataset and partitioned eighty percent of the player action trajectories for training, ten percent of the trajectories for validation, and the remaining ten percent of the trajectories for testing. In total two classifier types were created, where each classifier type has a play log variant based on the S-TLI and E-TLI play log instantiations. The first classifier type being a weighted KNN classifier and the second being our Hybrid Bayesian classifier type which is informed by a weighted KNN classifier as the likelihood probability function. From a baseline perspective we make use of a simple 'random' classifier which will at each step make a random prediction on the player's playstyle. In terms of our comparative baseline we again make use of the Ingram et al. approach [Ingram *et al.* 2022]. Two Ingram classifier variants are created based off the S-TLI and E-TLI play log instantiations. For the unsupervised Ingram classifiers we make use of the ground truth cluster labels generated in order to compute the performance of this model.

4.1.5 Special Case: Super Mario Bros Experimental Results

For the Super Mario Bros evaluation the intuition behind our choice of play log representation is that a lower level play log granularity in terms of the player interaction should better surface or distinguish the underlying prospective playstyles. Which in turn should bring about a stronger prediction accuracy performance. The average correct prediction percentage results obtained for the S-TLI and E-TLI evaluations are respectively shown in Figures 4.14 and 4.15. These results confirm the ability of our framework to operate and scale within a more complex domain using human player action trajectory data. Secondly, the difference in the average correct prediction percentage results between the S-TLI and E-TLI Hybrid classifiers confirm our intuition related to the player interaction play log granularity. The only difference between the S-TLI and E-TLI evaluations is this play log granularity, which results in a substantial difference in the playstyle classifiers' prediction accuracy.

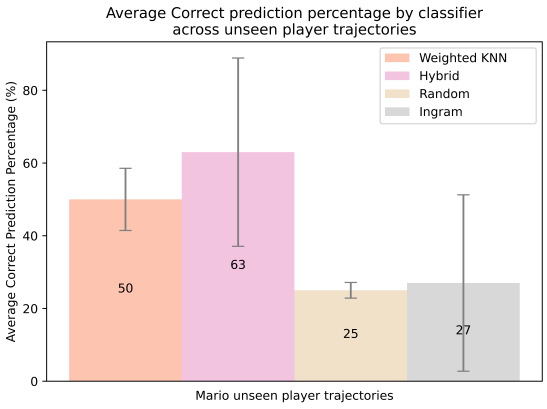


Figure 4.14: Mario S-TLI average correct prediction percentage by classifier across unseen player trajectories.

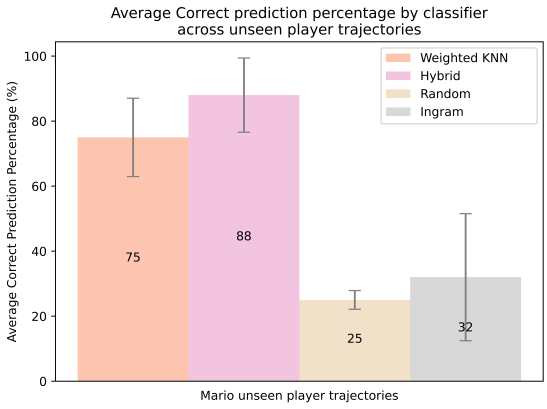


Figure 4.15: Mario E-TLI average correct prediction percentage by classifier across unseen player trajectories.

When comparing the weighted KNN classifier results to our Hybrid classifier we see a greater difference in average prediction performance, which we attribute to the respective stability scores obtained. As shown in Table 4.2, our Hybrid classifier's Bayesian belief update results in a better stability score than the Weighted KNN which more rapidly changes the playstyle prediction type due to the nature of the nearest neighbour search. The stability of the classifier is a count of how many times, during an episode, the classifier changes its playstyle type prediction (e.g. transitioning from a Monster Killer prediction to a Brave Treasure Hunter prediction). The lower the stability score, the more stable the classifier.

Table 4.2: Overall Average Stability Score by Classifier

Classifier Evaluation	**Average Stability Score**	
	S-TLI	***E-TLI***
Weighted KNN	0.196	0.080
Hybrid	**0.014**	**0.004**
Random	0.745	0.754
Ingram	0.040	0.161

In comparison to the MiniDungeons evaluation, the Mario evaluation better highlights the effect that the stability of the classifier has on the end prediction result. The Hybrid classifier's more gradual Bayesian belief update allows for the classifier output to be more certain about the playstyle under observation. It is important to note that the trade-off between the prediction accuracy and the model stability still exists. The MiniDungeons evaluation shows that a better stability does not necessary guarantee a better average prediction result. In the MiniDungeons case both the stability scores of the Ingram (TLI) and Bayes (TLI) were more stable in comparison to the Hybrid (TLI), but the average correct prediction percentage results were not better than the Hybrid (TLI). The strength of our Hybrid classifier and framework is again highlighted as our classifiers at each step received only the latest play log entry, whilst the Ingram classifiers received the latest play log entry as well as the previous temporal play log entries. Thus again confirming the success of the variable length pre-processing step. Furthermore, the Mario experiment highlights the robustness of our hybrid classifier to operate and scale within a more complex domain using real human player action trajectory data.

4.2 Learning to Assist Evaluation by Experiments

In this section we evaluate our method, discussed in Section 3.2, for creating our adaptive playstyle-aware companion agent. Our companion agent should be adaptive such that the action policy used should be tailored and complementary to the playstyle. We make use of Collaborative MiniDungeons, presented in Subsection 4.2.1, as our evaluation domain for this experiment. In order for our companion agent to be able to assist different kinds of playstyles we need a set of policies from which to choose from. We

begin this discussion related to the Companion Agent Policy Creation and reward function creation in Subsection 4.2.2. We then discuss the relevant experiment set-up and implementation detail in Subsection 4.2.3. In Subsection 4.2.4 we present the results and outcomes from the Companion Agent Action Selection evaluation.

4.2.1 Collaborative MiniDungeons

MiniDungeons, as described in Subsection 4.1.1, is a single player game. In order to evaluate our adaptive companion agent we extend MiniDungeons into a two-player game, as shown in Figure 4.16. In Collaborative MiniDungeons two agents traverse the dungeon interacting with the various level elements, as shown in Figures 4.16 and 4.17.

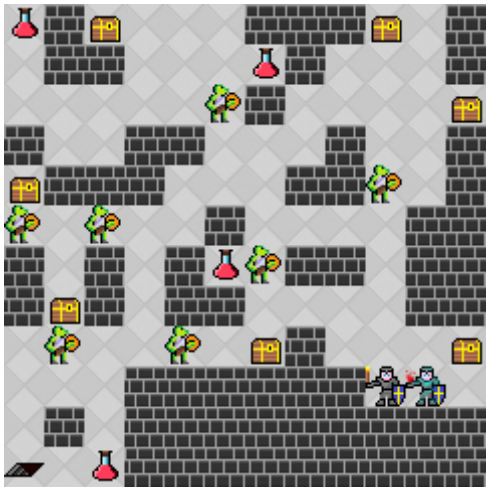


Figure 4.16: Collaborative MiniDungeons (Level 9)

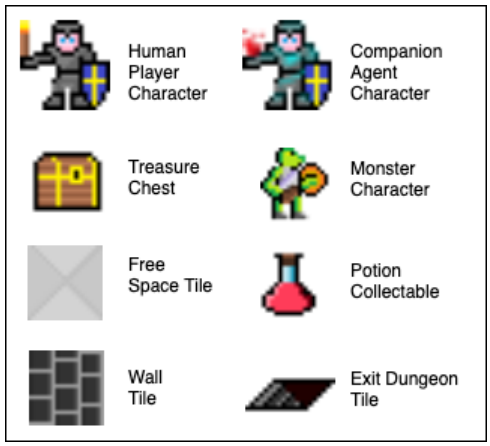


Figure 4.17: Collaborative MiniDungeons Legend

The two agents, in our case a human playstyle agent and a companion agent can work together to maximise the human player's playstyle objective. For example, a treasure centric player which purely collects treasure can be enabled to collect more treasure when the companion agent battles monster which guard treasure. Thus allowing the treasure centric player to collect more of the treasure in the collaborative setting in contrast to trying to complete the dungeon alone or where the companion agent does not assist the player strategy. The agents, actions, and observation dynamics are illustrated in Figure 4.18.

As illustrated in Figure 4.18 each agent has an action list with the respective input actions. Each agent must take their own action which is then passed to the MiniDungeons game environment. The game environment first applies the first agent's action (in our case Agent 1 is the human playstyle agent) accordingly. Thereafter, the second agent's action is applied (in our case Agent 2 is the companion agent). The actions within the collaborative MiniDungeons environment remains the same as the actions

available in the single player MiniDungeons (please see Subsection 4.1.1 for a recap regarding the actions available in MiniDungeons).

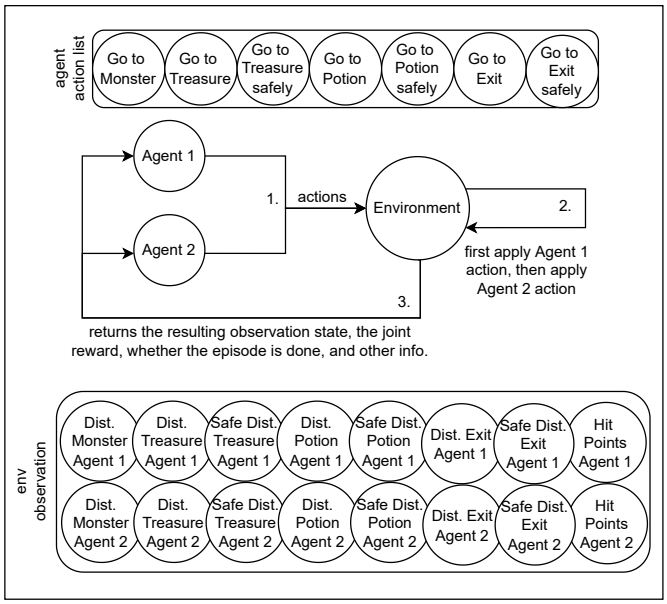


Figure 4.18: Collaborative MiniDungeons Gym Environment Dynamics, where the Agent 1 is the primary human proxy agent and Agent 2 is the companion agent.

The collaborative MiniDungeons gym environment returns: the resultant observation, or state, resulting from the first and second agent actions, the joint reward received for the agent actions pair, a boolean indicating whether the level episode is complete or not, and other info such as the latest play log. The resultant collaborative environment observation, or state, is represented as a list containing the first agent's hit points (HP) and distances from the first agent to their nearest world elements (distance to nearest monster, and the safe/unsafe distances to the nearest treasure, potion, and exit). The rest of the list contains the second agent's HP and respective distances.

The current build of Collaborative MiniDungeons will be contributed back to the original gym-md project repository. We create collaborative, two-player, versions of the standard eleven MiniDungeons levels created by Holmgard *et al.* [Holmgård *et al.* 2014c][Holmgård *et al.* 2014b]. We keep the layout of the levels the same, but include a secondary agent which starts the game next to the main character. The reason for creating two-player versions of the MiniDungeons levels created by Holmgard *et al.* is because these levels already sufficiently allow for different playstyles to express themselves. Furthermore, these levels are already a benchmark within the field of player modelling. The secondary agent in our experiments is our companion agent informed by our hybrid playstyle classifier, outlined in Section 3.1. Figure A.2, in Appendix A, illustrates the layouts for the respective eleven collaborative levels.

4.2.2 Companion Agent Policy Creation

We aim to find a mapping between a set of human proxy playstyles and the set of companion agent policies. For the human player proxy playstyles in Γ we make use of the same six rule-based agents, presented in Subsection 4.1.2, used as part of the Research Objective I MiniDungeons Experiment. The six rule-based agents are: *pure treasure hunter*, *brave treasure hunter*, *monster killer*, *heal and run*, *safe runner*, and *wreckless runner*.

For the agent policies in Π we make use of Deep Q-learning agents to represent the companion agent policies. For our discussion related to creating agent policies using Deep Q-learning, please see Subsection 3.2.2. In total seven agent policies are created to represent the companion agent policies in Π. The first companion agent is a mimic based adaptive agent which mimics the identified playstyle. A baseline random companion agent policy is created which performs a random action at every step in the episode. The remaining five companion agents are Deep-Q agents developed using different reward functions. Each reward function is shaped such that it returns different reward values which correspond to the respective objective which the playstyle aims to maximise. For example in the case of MiniDungeons, the reward function for a companion agent policy which favours treasure collection over monster kills will return a greater reward for treasure collection and a negative reward for monster kills.

The five Deep-Q agents developed are respectively called: *monster policy*, *monster with exit policy*, *treasure with kill policy*, *treasure policy*, and *potion policy*. The respective policy training graphs can be seen in Figure B.1, in Appendix B. Additionally, the respective policy reward function values can be found in Table B.1, in Appendix B. Regarding the agent playstyles the *monster policy* and *monster with exit policy* are both monster focused agents, with the difference being that the *monster with exit policy* is able to exit the dungeon. In contrast the *monster policy* will not exit the dungeon level on its own, but will rather delay taking the dungeon exit so that the human player can decide when to leave the dungeon. The *treasure with kill policy* and *treasure policy* agents are similar to the *brave treasure hunter* and *pure treasure hunter* such that both agents are treasure centric agents. The *treasure with kill policy* companion agent, like the *brave treasure hunter*, aims to collect treasure regardless of whether the treasure is guarded or not. The *treasure policy* companion agent, like the *pure treasure hunter*, only collects unguarded treasure. The *potion policy* companion agent focuses on collecting potions.

It is important to note that we are mainly concerned with determining the best response, or mapping, for a given playstyle in set Γ from the set of policies in Π, as defined in Subsection 3.2.1. Additionally, our method is not concerned with optimal policy creation for the policies in Γ and Π. For these reasons, we are only concerned that there exists these two sets Γ and Π. Furthermore, the sets Γ and Π may be disjoint, or they may be overlapping sets. For this experiment, we have made the sets Γ and Π differ in terms of the underlying implementation of the agent policies (rule-based vs. Deep Q-learning).

From a level allocation perspective the following Collaborative MiniDungeons levels were respectively assigned, through random assignment, to a train and test level sets. For this particular experiment, the train and test level sets are:

- L_{train} (md-collab-holmgard_0-v0, md-collab-holmgard_2-v0, md-collab-holmgard_6-v0, md-collab-holmgard_9-v0)
- L_{test} (md-collab-holmgard_1-v0, md-collab-holmgard_3-v0, md-collab-holmgard_4-v0, md-collab-holmgard_5-v0, md-collab-holmgard_7-v0, md-collab-holmgard_8-v0, md-collab-holmgard_10-v0).

For the training of the Deep-Q agents the L_{train} level set is used. Each agent policy was trained for ten-thousand episodes, and each episode was one-hundred steps long. The level for the episode is each time randomly selected from the L_{train} level set. The reason for the standardised configuration of the number of episodes alludes back to our method only being concerned that there exists these two agent policy sets Γ and Π. Furthermore, as mentioned we are not focused on creating optimal policies. From a network architecture perspective we have one input layer, three hidden layers, and an output layer, all of which are fully connected linear layers as shown in Figure 4.19. During training an Adam optimizer and the Mean-squared error was used. Furthermore, a reply memory of max $100,000$ size, with a batch size of 200 was used during the training. From a parameter perspective a learning rate of 0.01, a discount rate of 0.9, and a decreasing epsilon value relative to the number of games played is used. This architecture was iteratively developed and was chosen since it is sufficiently able to create a diverse set of companion agent policies based on the rewards functions defined (the respective policy reward function values can be found in Table B.1, in Appendix B).

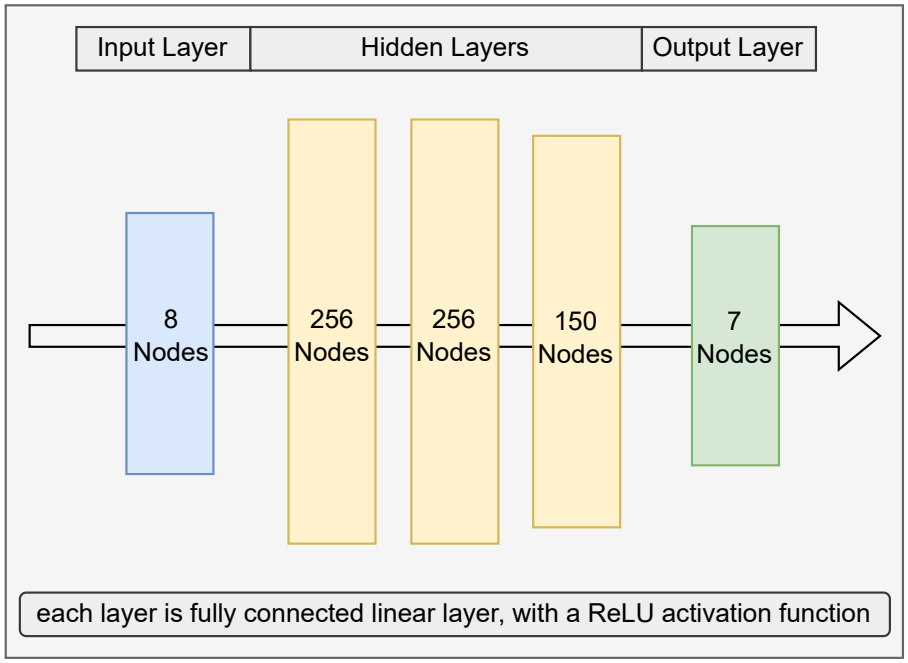


Figure 4.19: Companion Agent Polices Deep-Q Network architecture

The eight nodes in the input layer correspond to the MiniDungeons environment state representation. The observation of the environment is represented as a list of integers of length eight, where:

- index 0: Distance to the monster
- index 1: Distance to the treasure
- index 2: Distance to treasure (avoid monsters)
- index 3: Distance to potion
- index 4: Distance to potion (avoid monsters)
- index 5: Distance to the exit
- index 6: Distance to the exit (avoid monsters)
- index 7: Agent's physical strength (i.e. Hit Points (HP))

The seven nodes in the output layer correspond to the seven actions which an agent can take within the MiniDungeons gym environment. An action within the MiniDungeons environment is represented as a list containing seven floating point values, where

- index 0: Head to the monster
- index 1: Head to the treasure
- index 2: Head to the treasure (avoid monsters)
- index 3: Head to the potion
- index 4: Head to the potion (avoid monsters)
- index 5: Head to the exit
- index 6: Head to the exit (avoid monsters)

The highest action value in the action list is first attempted, if this action cannot be taken in the given state then the next highest action is selected and so on. These actions are encoded by the MiniDungeons gym environment. Each action heads to the respective nearest target (monster, treasure, potion, or exit). The path taken towards the given target can either be safe or unsafe. A safe path does not contain monsters, whereas an unsafe path does encounter monsters. We have contributed quite extensively to the MiniDungeons gym environment repository's readme[8], please see the readme and the respective function documentation for more detail.

In practise, the usefulness of the companion agent is determined or judged by the

[8]https://github.com/ganyariya/gym-md

human player. We recognise this is a limitation of our experimental paradigm, since we have not incorporated real human player feedback during the evaluation of our companion agent policies. The assumption made by our method is that for each playstyle a reward function exists which encapsulates what the human playstyle finds rewarding. Existing literature shows that a diverse set of playstyle proxy agents can successfully be created and represent human players in a given game. For this reason and since user testing is not in the scope of this method, we make use of playstyle proxy agents with defined reward functions to sufficiently represent the real human player feedback.

What each human player values, or finds rewarding is different. Human players find different parts of a given game rewarding. For example, a monster experience centric player values battling monsters, a treasure focused player playstyle values looting treasure and so on. Therefore, the environment reward function used should be adaptive and is dependent on the human playstyle that the companion agent is assisting. Thus in our experiment we dynamically adjust the environment reward function based on the human playstyle agent under test. For example, when the *safe_runner* playstyle agent is loaded the corresponding reward function values is used to determine the joint agent reward. The set of human playstyle proxy reward functions, ϖ, is summarised in Table 4.3.

Table 4.3: Human Proxy Agent Reward Function Values

Human Proxy Agent	**Reward Values**
MONSTER_KILLER	{ “TURN”: 1, “EXIT”: 5, “KILL”: 50, “TREASURE”: -5, “POTION”: 1, “DEAD”: -25 }
SAFE_RUNNER	{ “TURN”: 1, “EXIT”: 50, “KILL”: -5, “TREASURE”: -5, “POTION”: 5, “DEAD”: -50 }
WRECKLESS_RUNNER	{ “TURN”: 1, “EXIT”: 50, “KILL”: 1, “TREASURE”: 1, “POTION”: 1, “DEAD”: -15 }
HEAL_AND_RUN	{ “TURN”: 1, “EXIT”: 5, “KILL”: -5, “TREASURE”: -5, “POTION”: 50, “DEAD”: -25 }
PURE_TREASURE_HUNTER	{ “TURN”: 1, “EXIT”: 5, “KILL”: -5, “TREASURE”: 50, “POTION”: 1, “DEAD”: -25 }
BRAVE_TREASURE_HUNTER	{ “TURN”: 1, “EXIT”: 5, “KILL”: 30, “TREASURE”: 50, “POTION”: 1, “DEAD”: -25 }

Regarding the Learning to Assist problem definition, in Subsection 3.2.1, we have addressed the creation of the respective human proxy and companion agent policy sets, Γ and Π. Additionally for each playstyle in Γ we have defined the corresponding reward functions for the set ϖ. In Subsection 4.2.3 we discuss our process for creating our

mapping function, Λ, which produces a mapping for every playstyle $p \in \Gamma$ to an agent policy $\pi \in \Pi$ such that the mapping is complementary.

4.2.3 Human Proxy Playstyle and Companion Agent Policy Mapping Process

In order to create the playstyle to companion agent policy mapping we need to evaluate each agent Playstyle-Policy pair and record the respective reward value outcomes. We make use of the L_{train} Collaborative MiniDungeons level set to create the initial mapping. It is important to note that the mapping is performed offline and can be an iterative process to determine the best mapping pairs. We record the respective reward value outcomes for each Agent Playstyle-Policy pair. The pairing with the highest average joint-reward is selected as the Playstyle-Policy mapping. Refer to Algorithm 5, in Appendix B, which illustrates this human playstyle and companion agent policy mapping process. Figure 4.20 illustrates the human proxy playstyle and companion agent policy mapping outcome. This mapping outcome is what our adaptive companion agent uses at runtime to determine the best policy response for the classified playstyle.

Table 4.4 summarises the outcome from our mapping process. For each level in the training set, the playstyle companion agent pairing plays the same level ten times. The resultant episodic reward values are then averaged across the ten level runs and then the average reward value across the training levels are calculated. The playstyle-policy match-up with the highest average joint-reward is made bold in Table 4.4.

Table 4.4: Mapping process summary of the overall average joint reward for each playstyle-policy match-up

		Companion Agent Policy						
		random	mimic	monster_w_exit_policy	treasure_w_kill_policy	monster_policy	treasure_policy	potion_policy
Playstyle	SAFE_RUNNER	-7.125	**-1.75**	-2	-21.125	-57.8	-9.1	-1.975
	WRECKLESS_RUNNER	15.774	16.0	**18.0**	17.15	6.25	15.9	16.950
	MONSTER_KILLER	206.924	226.975	**253.5**	214.675	239.950	179.0	159.15
	HEAL_AND_RUN	36.699	**60.25**	55.425	13.900	-1.949	33.025	46.95
	PURE_TREASURE_HUNTER	78.45	58.5	80.775	**124.15**	59.525	77.525	64.149
	BRAVE_TREASURE_HUNTER	289.824	291.7	**309.825**	296.95	211.55	293.125	256.875

It is interesting to see that there is not a one-to-one mapping between the proxy agent and companion agent sets. Out of the seven companion agent policies created only three are paired to the proxy agent playstyle set. It is important to note that each pairing, as illustrated in Figure 4.20, is the pairing with the highest average joint-reward (as summarised in Table 4.4).

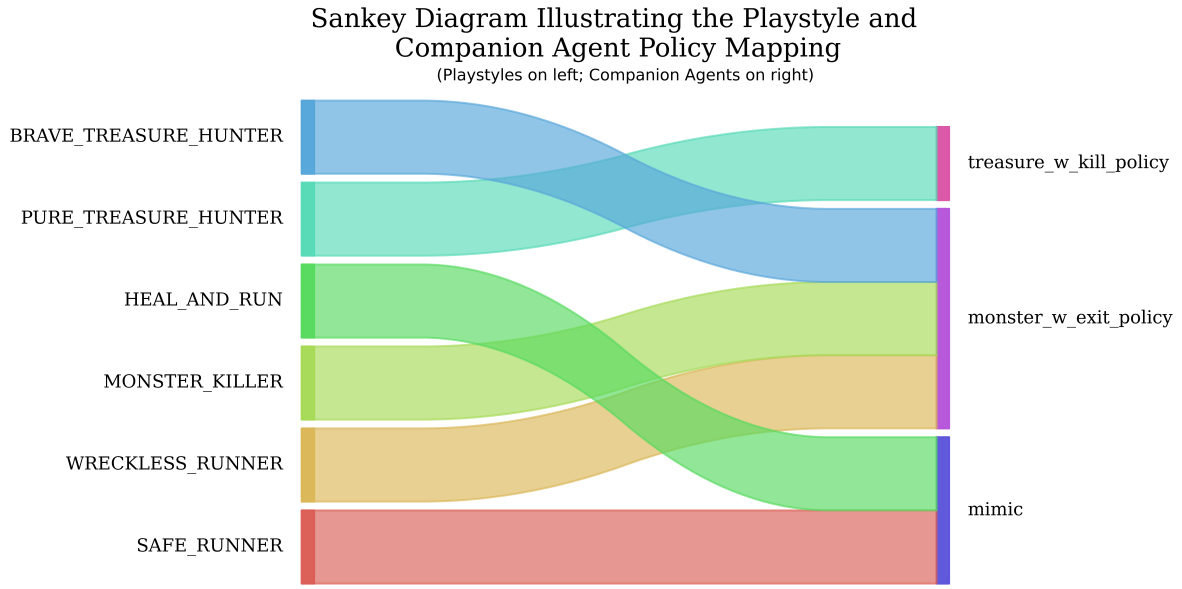


Figure 4.20: Human Proxy Playstyle and Companion Agent Policy Mapping

Both the *heal and run* and *safe runner* playstyles are mapped to the *mimic* based adaptive companion agent. This pairing makes sense since both the *heal and run* and *safe runner* are playstyles with quite singular goals. The *heal and run* playstyle is only concerned with getting through the dungeon as quick as possible only collecting potions. Similarly for the *safe runner* playstyle which values exiting the dungeon as quick as possible using the shortest safe route, containing no enemies. A mimic based style of "copy as I do" makes sense, the player does not value or want their companion agent deviating from their respective plans. In this case the companion should focus on aligning and do as the player would. The mimic based companion agent policy is an adaptive agent which changes its logic based on the observed playstyle. The mapping of two of the six playstyles to the mimic agent is an important outcome. The fact that not all six proxy playstyles map it to mimic agent strongly suggests that a mimic based approach is not the overall optimal complementary strategy. If simply mimicking the human player was the optimal average complementary strategy we would have expected to see majority, if not all six, of the proxy human playstyles mapping to the mimic based companion agent policy. Companion agent policies which performs identical actions to the human player might bring the human player closer to their respective goal, but a different action may better complement or advance the player strategy and goal [Scott and Khosmood 2017].

The *monster killer* playstyle as well as the *brave treasure hunter* playstyle both mapped to the *monster with exit policy* companion agent policy. This mapping makes sense since both the *monster killer* and the *brave treasure hunter* playstyles gain positive reward values for battling monsters. A complementary companion agent strategy such as the *monster with exit policy* also prioritises monster battling over other objectives in MiniDungeons. In the case of the *monster killer* playstyle, the more monsters slain the better, the *monster with exit policy* complements this goal well. The *brave treasure*

hunter playstyle aims to collect as much, free or guarded, treasure as possible. The *monster with exit policy* complements this goal well, such that the *monster with exit policy* battles monsters which may guard treasure, thus freeing up more treasure to be collected by the agent team. The *wreckless runner* playstyle is also mapped to the *monster with exit policy*. The *wreckless runner* aims to exit the dungeon as fast as possible using the shortest path out. The *monster with exit policy* complements the *wreckless runner* goal by clearing out any monsters along the shortest path to the exit. Thus complementing the strategy better than a pure mimic style, since the *monster with exit policy* decreases the risk of the human player dying due to trying to follow an unsafe shortest path filled with monsters, whereas a mimic style agent would simply follow the human player possibly also dying in the process.

The *pure treasure hunter* playstyle only collects unguarded treasure, and will not engage in enemy battles. The *pure treasure hunter* playstyle is mapped to the *treasure with kill policy* which is also a treasure centric agent but the *treasure with kill policy* agent has the ability to engage in enemy battles. The *pure treasure hunter* playstyle and *treasure with kill policy* are complementary such that the team prioritises treasure collection, the *treasure with kill policy* battles monsters which may guard treasure, thus freeing up more treasure to be collected by the agent team.

It is important to note that little emphasis should be placed on the specific agent policies (which are somewhat arbitrary for both the agent policy sets, Γ and Π). Our method is mainly concerned with determining the best response, or mapping, for a given playstyle in set Γ from the set of policies in Π. Additionally, our method is not concerned with optimal policy creation for the policies in Γ and Π. For these reasons, we are only concerned that there exists these two sets Γ and Π. The core takeaway is the general methodology enabling the creation of the human proxy playstyle and companion agent policy mapping. The adaptive playstyle-aware companion agent's action selection, presented in Subsection 4.2.4, makes use of this human proxy playstyle to companion agent policy mapping to respond accordingly to the predicted playstyle.

4.2.4 Companion Agent Action Selection

The companion agent action selection is dependent on the existence of the playstyle classification model which can at every step, within a Minidungeons level, classify the unseen play log entries of an observed player to a playstyle, as defined in Equation 3.1. Additionally, the companion agent action selection is dependent on the existence of the Playstyle-Policy Mapping, defined in Subsection 3.2.3. Our playstyle-aware adaptive companion agent makes use of the companion agent action selection process defined in Algorithm 4. For a given step within an episode, the companion agent makes use of the playstyle classifier to classify the observed player. The Playstyle-Policy Mapping is then used to find the mapped policy response. The given state along with the mapped policy is then used to determine the respective action to take.

The playstyle classification module of our adaptive companion agent is the Hybrid Bayesian Supervised Learning classifier, as presented in Section 3.1. From a play log

instantiation perspective we make use of the Tactic level information (TLI) play log representation, presented in subsection 4.1.2. The TLI based Hybrid Bayesian Supervised Learning classifier was chosen, for the adaptive agent, since it has the best performance and a better balance in terms of the stability score when compared to the other classifier and play log representation combinations.

For each level in the L_{test} level set for Collaborative MiniDungeons, we evaluate each proxy playstyle and companion agent policy combination and record the respective joint reward obtained across the level runs, where each level was played ten times. We make use of Algorithm 5, in Appendix B, and alter the input level set to be the L_{test} level set, and we include our adaptive companion agent into our companion agent policy set Π. Thus allowing us to obtain the respective joint reward using the L_{test} level set and to evaluate our adaptive companion agent against the existing companion agent policies in Π. After rerunning the mapping process on these new input parameters we expect to see our companion agent chosen more often than not as the most complementary or suitable companion agent for each respective playstyle. Additionally, we expect to see our adaptive playstyle-aware companion obtain a higher overall average joint reward when compared to the agents who do not adapt their action based on the playstyle. The first reason for this is that on average our playstyle-aware companion agent should better adapt over time as the playstyle classifier tracks the playstyle belief. Secondly our playstyle-aware companion should respond most suitably to the player because of the Playstyle-Policy mapping. Figures 4.21 and 4.22 summarise the outcome from our L_{test} set evaluation of our playstyle adaptive companion agent.

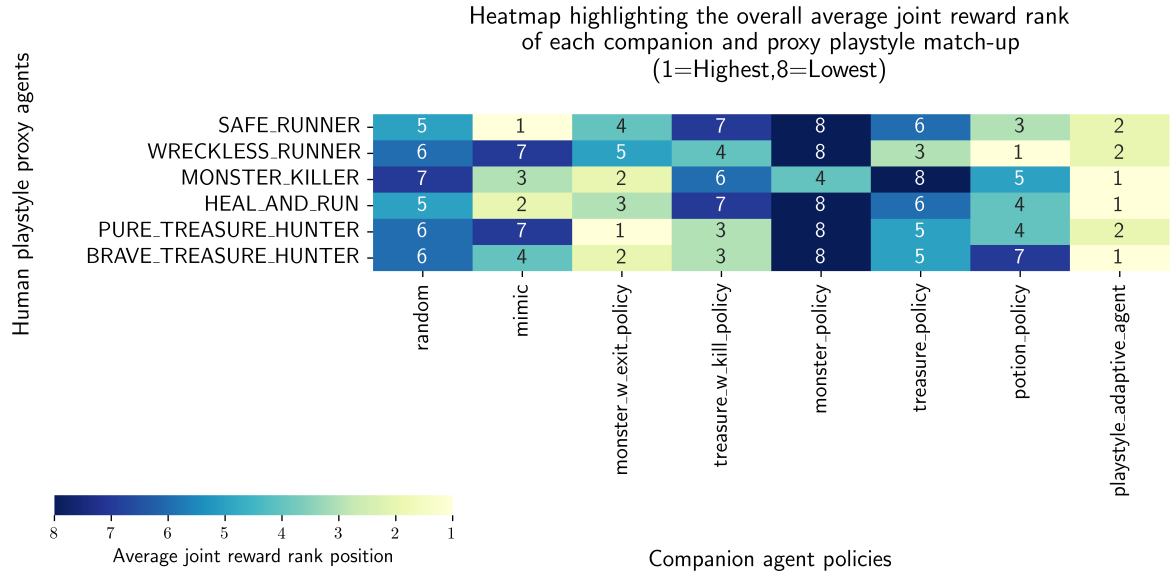


Figure 4.21: Heatmap highlighting the overall average joint reward ranking of each companion and proxy playstyle match-up

Figure 4.21 summarises the overall average joint reward ranking of each companion and proxy playstyle match-up. In other words, we take each playstyle-policy match-up and rank the average joint reward obtained. Figure 4.22 shows the actual average joint

reward obtained overlaid on top of the ranking view. Additionally Figure 4.22 highlights the difference between the top two ranked values for each playstyle.

In Figure 4.21 we see that for each playstyle our adaptive companion agent is either ranked first or second in terms of the average joint reward obtained. This suggests that overall our companion agent is the most consistent across the playstyles, bringing about either the best or second best average joint reward. The use of the joint reward as a measure is suitable since the reward used during each evaluation corresponds directly to the internal motivation behind each playstyle, as shown in Table 4.3. In other words, the reward function which each playstyle and agent policy aims to maximises is the reward function which intrinsically represents what the player values and possibly enjoys. Another important observation is that no other companion agent policy is as consistent across all the proxy human playstyles in comparison to our adaptive companion agent. For example, the potion companion agent policy is ranked first for the *wreckless runner* playstyle, but ranks seventh best for the *brave treasure hunter* and fifth for the *monster killer* playstyle. Thus highlighting the inability of the potion companion agent policy to generalise and adapt to different playstyles. Despite obtaining the higher joint reward for the *wreckless runner* playstyle, the potion companion agent policy is too tailored and specific to perform well with any other playstyle. Thus highlighting the importance of adaptability in game AI in order to cater for a diverse player base.

For each playstyle our adaptive companion agent is either ranked first or second in terms of the average joint reward obtained. It is important to inspect and discuss the difference in terms of the average joint reward value between ranks one and two, as shown in Figure 4.22.

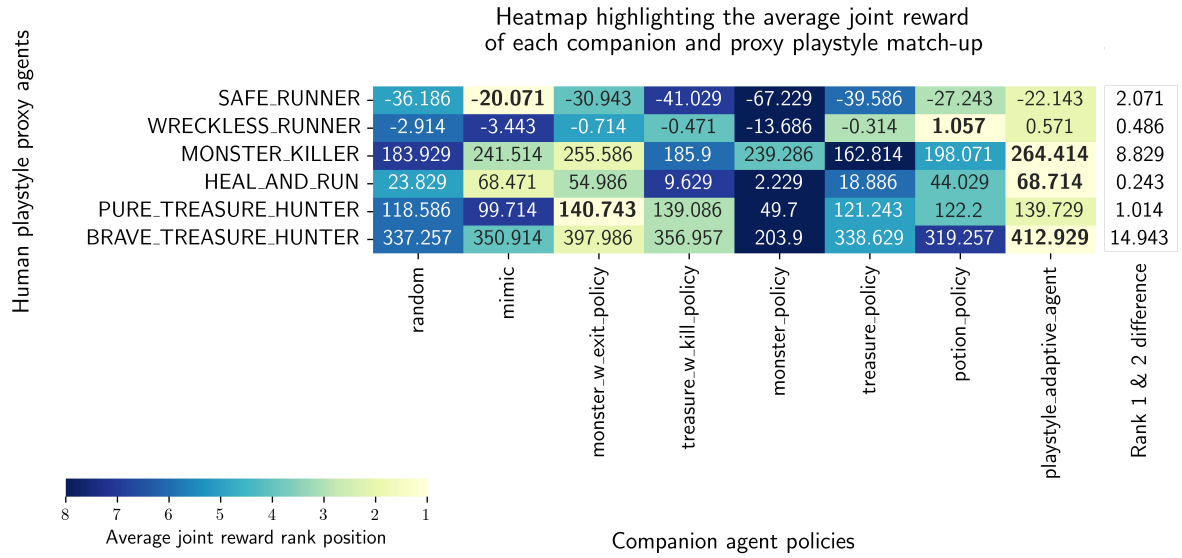


Figure 4.22: Heatmap highlighting the overall average joint reward of each companion and proxy playstyle match-up

Our playstyle adaptive agent obtains the greatest average joint reward value for the

monster killer, *heal and run*, and *brave treasure hunter* playstyles, with a respective $1^{st} - 2^{nd}$ rank difference, in average joint reward, of 8.829, 0.243, and 14.943. Secondly, our playstyle adaptive agent obtains the next greatest average joint reward value for the *safe runner*, *wreckless runner*, and *pure treasure hunter* playstyles, with a respective $1^{st} - 2^{nd}$ rank difference, in average joint reward, of 2.071, 0.486, and 1.014. The $1^{st} - 2^{nd}$ rank difference in average joint reward highlights the strength, success, and generalisability of our adaptive companion agent. On average in the situations when our adaptive companion agent is ranked first, the $1^{st} - 2^{nd}$ rank difference is larger. In other words, our adaptive companion agent outperforms every other companion agent policy by a larger margin. Additionally, when we look at the situations when our adaptive companion agent is ranked second, on average the $1^{st} - 2^{nd}$ rank difference is small. Thus, implying that our companion agent is only marginally behind the first ranked companion agent.

In summary, when our adaptive agent is ranked first it is the optimal choice by a larger margin, and in the cases when our adaptive agent is ranked second it is by a small marginal amount. This is an important outcome, highlighting the adaptability of our companion agent in catering for a diverse player base. Meaning that the advantage that our adaptive companion agent has, when compared to the non-personalised cases, is that the adaptive companion agent is able to bring about a greater player experience when using the playstyle specific reward function as a proxy for what the players find rewarding. In Chapter 5 we discuss how we would like to further improve the adaptive capabilities of our playstyle-aware companion agent.

Chapter 5

Conclusion

In this work, we explored the problem of creating an adaptive companion agent that is able to assist a human in solving collaborative games, whilst taking into account their playstyle. Creating a playstyle-aware adaptive agent firstly requires a mechanism for correctly classifying or identifying the player style, before attempting to assist the player with a given task. We divided our research problem into two sub-problems: *'Learn to identify playstyles'* and *'Learning to Assist'*. To solve we presented a method which can enable the real time in-game playstyle classification of players. We contribute a hybrid probabilistic supervised learning framework, using Bayesian Inference informed by a K-Nearest Neighbours based likelihood, that is able to classify players in real time at every step within a given game level using only the latest player action or state observation. Furthermore we outperform comparative baselines whilst using only the latest player observation to make our prediction. Our playstyle classification experiment highlights the success of our framework in a complex human player setting and the effect the play log representation has on the prediction's accuracy, stability, and generalisability. As part of future work we plan to use our playstyle classification framework to adapt the gameplay experience in a meaningful way. We would also like to apply our work in a competitive online game setting in order to illustrate the potential use and benefit to the e-sports community from a player strategy analysis and game analytics perspective.

Companion agents should be designed in such a way that the game AI behaviour aligns with the player strategy and intention [Bakkes *et al.* 2012]. To solve sub-problem II we created a method which uses the playstyle classification module to assist the identified playstyle with the most suitable expected action. In other words, the companion agent should take actions which complement the player such that the player is able to achieve their playstyle specific goal. Furthermore, the companion agent should be adaptive such that the action policy used should be tailored and complementary to the playstyle. We use Collaborative MiniDungeons as our evaluation domain for this experiment. The core problem which we aimed to solve is the problem of determining the best, or more complementary, action response for a given playstyle. We define this problem as the 'Learning to Assist' problem: where given a set of agent policies, which can play a given game, determine the best policy which best complements the observed playstyle. We solve this through the creation of a playstyle-policy mapping function, which maps each human playstyle proxy to a complementary companion agent policy.

For a given step within an episode, the companion agent makes use of the playstyle classifier to classify the observed player. The Playstyle-Policy Mapping is then used to find the mapped policy response. The given state, along with the mapped policy, is then used to determine the respective action to take. Our companion agent is adaptive and able to collaborate with various playstyles, achieving the greatest overall average joint-reward. Thus providing the most consistent and reliable collaborative experience in comparison to the non-adaptive companion agent baselines. Our companion agent method is successful in the sense that our adaptive playstyle companion agent is able to bring about a smoother player experience when using the playstyle specific reward function as a proxy for what the players find rewarding. This is an important outcome, highlighting the adaptability of our companion agent in catering for a diverse player base.

From a scaling perspective, we would like to further evaluate our companion agent method within a more complex multi-agent collaborative domain, such as Hanabi [Bard *et al.* 2020] or Geometry Friends [Prada *et al.* 2015]. Furthermore, we would like to perform a user study wherein human players would collaborate and assess our adaptive companion agent providing valuable feedback on the Human-AI collaborative experience. We also plan to address the reward-function requirement of our companion agent framework. A current limitation of our companion agent framework is that a reward function should be defined along side each playstyle. Creating a reward function which represents a given human behaviour is challenging in itself. For this reason, currently we make us of playstyle agents to represent human players. As discussed, in our background (Chapter 2), a diverse set of playstyle proxy agents can successfully be created and represent human players in a given game. Furthermore, in the situation where no historic player data exists, playstyle proxy agents with defined reward functions are sufficient. However, as real human player play log data becomes available inverse reinforcement learning techniques can be explored to learn a given playstyle reward function instead of hand-crafting the playstyle reward function [Arora and Doshi 2021].

References

[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.

[Arendse *et al.* 2022] Lindsay John Arendse, Branden Ingram, and Benjamin Rosman. Real time in-game playstyle classification using a hybrid probabilistic supervised learning approach. In Anban Pillay, Edgar Jembere, and Aurona Gerber, editors, *Artificial Intelligence Research. SACAIR 2022. Communications in Computer and Information Science*, pages 60–77, Cham, 2022. Springer Nature Switzerland.

[Arora and Doshi 2021] Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500, 2021.

[Bakkes *et al.* 2012] S. Bakkes, P. Spronck, and G. V. Lankveld. Player behavioural modelling for video games. *Entertainment Computing*, 3:71–79, 2012.

[Bard *et al.* 2020] Nolan Bard, Jakob N. Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H. Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, Iain Dunning, Shibl Mourad, Hugo Larochelle, Marc G. Bellemare, and Michael Bowling. The hanabi challenge: A new frontier for AI research. *Artificial Intelligence*, 280:103216, 2020.

[Bishop 2006] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[Bolstad and Curran 2016] William Bolstad and James Curran. *Introduction to Bayesian Statistics, 3rd Edition*. Wiley, 2016.

[Bontchev 2016] Boyan Bontchev. Holistic player modeling for controling adaptation in video games. *Proceedings of the 14th International Conference e-Society 2016*, 2016.

[Bouquet *et al.* 2021] Elizabeth Bouquet, Ville Mäkelä, and Albrecht Schmidt. Exploring the design of companions in video games. In *Proceedings of the 24th International Academic Mindtrek Conference*, Academic Mindtrek '21, pages 145–153, New York, NY, USA, 2021. Association for Computing Machinery.

[Brockman *et al.* 2016] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[Certicky *et al.* 2019] Martin Certicky, Michal Certicky, Peter Sincak, Gergely Magyar, Ján Vaščák, and Filippo Cavallo. Psychophysiological indicators for modeling user experience in interactive digital entertainment. *Sensors*, 19, 2019.

[Charles and Black 2004] Darryl Charles and Michaela Black. Dynamic player modelling: A framework for player-centred digital games. *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, 2004.

[Cruz and Uresti 2018] Christian Arzate Cruz and Jorge Adolfo Ramirez Uresti. Hrlb2: A reinforcement learning based framework for believable bots. *Applied Sciences*, 8(12), 2018.

[Dalianis 2018] Hercules Dalianis. *Evaluation Metrics and Evaluation*, pages 46–48. Springer International Publishing, 2018.

[Emmerich *et al.* 2018] Katharina Emmerich, Patrizia Ring, and Maic Masuch. I'm Glad You Are on My Side: How to Design Compelling Game Companions. In *CHI PLAY '18*, 2018.

[Freedman and Zilberstein 2018] R. Freedman and S. Zilberstein. Roles that plan, activity, and intent recognition with planning can play in games. In *AAAI Workshops*, 2018.

[Friedman and Schrum 2019] Adina Friedman and Jacob Schrum. Desirable behaviors for companion bots in first-person shooters. *2019 IEEE Conference on Games (CoG)*, pages 1–8, 2019.

[Ghosh *et al.* 2020] Ahana Ghosh, Sebastian Tschiatschek, Hamed Mahdavi, and Adish Singla. *Towards Deployment of Robust AI Agents for Human-Machine Partnerships*, 2020.

[Gonzalez-Duque *et al.* 2021] Miguel Gonzalez-Duque, Rasmus Berg Palm, and Sebastian Risi. Fast game content adaptation through bayesian-based player modelling. In *2021 IEEE Conference on Games (CoG)*, pages 01–08, 2021.

[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.

[Guzdial and Riedl 2016] Matthew Guzdial and Mark O Riedl. Game level generation from gameplay videos. In *Proceedings of the Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 12, page 44–50, 2016.

[Hernandez-Leal *et al.* 2016] Pablo Hernandez-Leal, Matthew E. Taylor, Benjamin Rosman, Luis Enrique Sucar, and Enrique Munoz de Cote. Identifying and tracking switching, non-stationary opponents: A bayesian approach. In *AAAI Workshop: Multiagent Interaction without Prior Coordination*, 2016.

[Holmgård *et al.* 2014a] Christoffer Holmgård, 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.

[Holmgård *et al.* 2014b] Christoffer Holmgård, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis. Generative agents for player decision modeling in games. In *FDG*, 2014.

[Holmgård *et al.* 2014c] Christoffer Holmgård, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis. Personas versus clones for player decision modeling. In Yusuf Pisan, Nikitas M. Sgouros, and Tim Marsh, editors, *Entertainment Computing – ICEC 2014*, pages 159–166. Springer, 2014.

[Holmgård *et al.* 2021] Christoffer Holmgård, Antonios Liapis, Julian Togelius, and Georgios Yannakakis. Monte-carlo tree search for persona based player modeling. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 11(5):8–14, 2021.

[Hu *et al.* 2020] Hengyuan Hu, Adam Lerer, Alex Peysakhovich, and Jakob Foerster. "other-play" for zero-shot coordination. *Thirty-seventh International Conference on Machine Learning (ICML) 2020*, 2020.

[Hunter 2007] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[Iawasaki and Hasebe 2021] Yu Iawasaki and Koji Hasebe. Identifying playstyles in games with neat and clustering. In *2021 IEEE Conference on Games (CoG)*, pages 1–4, 2021.

[Ingram *et al.* 2022] Branden Ingram, Benjamin Rosman, Clint van Alten, and Richard Klein. Play-style identification through deep unsupervised clustering of trajectories. In *2022 IEEE Conference on Games (CoG)*, pages 393–400, 2022.

[Ingram 2021] Branden Ingram. Generating tailored advice in video games through play-style identification and player modelling. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 17(1):228–231, 2021.

[Iwasaki and Hasebe 2022] Yu Iwasaki and Koji Hasebe. A framework for generating playstyles of game ai with clustering of play logs. In *Proceedings of the 14th International Conference on Agents and Artificial Intelligence - Volume 3: ICAART,*, pages 605–612. INSTICC, SciTePress, 2022.

[Jadhav *et al.* 2020] Nachiket Jadhav, Aniket Matodkar, Anish Mandhare, and Sujata Bhairnallykar. Use of artificial intelligence and machine learning in games. *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)*, 11(2), 2020.

[Kaelbling *et al.* 1996] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. *Reinforcement Learning: A Survey*, volume 4. AI Access Foundation, 1996.

[Kline and Arlidge 2003] Stephen Kline and Avery Arlidge. *Online Gaming as Emergent Social Media: A Survey*, 2003.

[Kodinariya and Makwana 2013] Trupti Kodinariya and Prashant Makwana. Review on determining of cluster in k-means clustering. *International Journal of Advance Research in Computer Science and Management Studies*, 1:90–95, 2013.

[Lazzaro 2004] Nicole Lazzaro. Why we play games: Four keys to more emotion without story. *Game Dev Conf*, 2004.

[Lou 2017] Harbing Lou. *AI in Video Games: Toward a More Intelligent Game*, 2017. harvard.edu blog, Special Edition on Artificial Intelligence.

[Luo *et al.* 2018] Zijin Luo, Matthew Guzdial, Nicholas Liao, and Mark O. Riedl. Player experience extraction from gameplay video. In *AIIDE*, 2018.

[Macindoe *et al.* 2012] Owen Macindoe, Leslie Pack Kaelbling, and Tomas Lozano-Perez. POMCoP: Belief Space Planning for Sidekicks in Cooperative Games. In *AIIDE*, 2012.

[Mahlmann *et al.* 2010] Tobias Mahlmann, Anders Drachen, Julian Togelius, Alessandro Canossa, and Georgios Yannakakis. Predicting player behavior in tomb raider: Underworld. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, CIG2010*, pages 178 – 185, 2010.

[Makantasis *et al.* 2019] Konstantinos Makantasis, Antonios Liapis, and Georgios N. Yannakakis. From pixels to affect: A study on games and player experience. In *8th International Conference on Affective Computing and Intelligent Interaction (ACII)*, 2019.

[Malone 1980] T.W. Malone. What makes things fun to learn? heuristics for designing instructional computer games. In *Symposium on Small Systems*, 1980.

[Marsland 2014] Stephen Marsland. *Machine Learning: An Algorithmic Perspective, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2014.

[Mitchell 1997] Tom M Mitchell. *Machine learning*. McGraw-hill, 1997.

[Mnih and et al. 2015] Volodymyr Mnih and et al. Human-level control through deep reinforcement learning. In *Nature*, 2015.

[Nguyen *et al.* 2011] Truong-Huy Nguyen, David Hsu, Wee-Sun Lee, Tze-Yun Leong, Leslie Kaelbling, Tomas Lozano-Perez, and Andrew Grant. Capir: Collaborative action planning with intention recognition. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 7(1):61–66, 2011.

[Normoyle and Jensen 2021] Aline Normoyle and Shane Jensen. Bayesian clustering of player styles for multiplayer games. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 11(1):163–169, 2021.

[Pedersen *et al.* 2009] Chris Pedersen, Julian Togelius, and Georgios Yannakakis. Modeling player experience in super mario bros. In *CIG2009 - 2009 IEEE Symposium on Computational Intelligence and Games*, pages 132 – 139, 2009.

[Pedregosa and et al. 2011] F. Pedregosa and et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2011.

[Prada *et al.* 2015] R. Prada, P. Lopes, J. Catarino, J. Quitério, and F. S. Melo. The geometry friends game ai competition. In *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 431–438, 2015.

[Risi and Preuss 2020] Sebastian Risi and Mike Preuss. From chess and atari to starcraft and beyond: How game ai is driving the world of ai. In *Künstliche Intelligenz*, pages 1610–1987, 2020.

[Roberts *et al.* 2007] David L. Roberts, Christina R. Strong, and Charles Lee Isbell. Estimating player satisfaction through the author's eyes. In *Proceedings of the Second Workshop on Optimizing Player Satisfaction*, 2007.

[Russell and Norvig 2010] Stuart Russell and Peter Norvig. Artificial Intelligence: A Modern Approach, 3rd Edition. pages 161–195. Prentice Hall, 2010.

[Scirea 2020] Marco Scirea. Adaptive puzzle generation for computational thinking. *International Conference on Human-Computer Interaction (HCII 2020: HCI in Games)*, pages 471–485, 2020.

[Scott and Khosmood 2017] Gavin Scott and Foaad Khosmood. *Complementary Companion Behavior in Video Games*. Master's thesis, School of Computer Science, California Polytechnic State University, San Luis Obispo, https://digitalcommons.calpoly.edu/theses/1744, 2017.

[Siqueira *et al.* 2018] Elton Siqueira, Thiago Santos, Carla Castanho, and Ricardo Jacobi. Estimating player experience from arousal and valence using psychophysiological signals. In *2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2018.

[Sombat *et al.* 2012] Wichit Sombat, Philipp Rohlfshagen, and Simon M. Lucas. Evaluating the enjoyability of the ghosts in ms pac-man. *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 379–387, 2012.

[Sutton and Barto 2018] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[Černý 2015] Martin Černý. Sally : Creating a Likeable and Competent AI Sidekick for a Videogame. In *AAAI Publications, Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.

[Togelius *et al.* 2006] Julian Togelius, Renzo De Nardi, and Simon M. Lucas. Making racing fun through player modeling and track evolution. In *Proceedings of the SAB'06 Workshop on Adaptive Approaches for Optimizing Player Satisfaction in Computer and Physical Games*, 2006.

[Tomai and Flores 2014] E. Tomai and Roberto Flores. Adapting in-game agent behavior by observation of players using learning behavior trees. In *FDG*, 2014.

[Tremblay and Verbrugge 2013] Jonathan Tremblay and Clark Verbrugge. Adaptive Companions in FPS Games. In *FDG*, 2013.

[Tychsen and Canossa 2008] Anders Tychsen and Alessandro Canossa. Defining personas in games using metrics. In *Future Play*, 2008.

[Valls-Vargas *et al.* 2015] Josep Valls-Vargas, Santiago Ontañón, and Jichen Zhu. Exploring player trace segmentation for dynamic play style prediction. In *AIIDE*, 2015.

[Waskom 2021] Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6, 2021.

[Xia *et al.* 2020] B. Xia, X. Ye, and A. O. M. Abuassba. Recent research on ai in games. In *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pages 505–510, 2020.

[Yannakakis and Hallam 2004] Georgios Yannakakis and John Hallam. Evolving opponents for interesting interactive computer games. In *8th International Conference on the Simulation of Adaptive Behavior (SAB'04)*, 2004.

[Yannakakis and Hallam 2007] Georgios N. Yannakakis and John Hallam. Capturing player enjoyment in computer games. In *Advanced Intelligent Paradigms in Computer Games*. Springer, 2007.

[Yannakakis and Hallam 2009] Georgios N. Yannakakis and John Hallam. Real-time game adaptation for optimizing player satisfaction. *IEEE Transactions on Computational Intelligence and AI in Games*, 1:121–133, 2009.

[Yannakakis and Togelius 2015] G. N. Yannakakis and J. Togelius. A panorama of artificial and computational intelligence in games. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(4):317–335, 2015.

[Yannakakis and Togelius 2018] Georgios N. Yannakakis and Julian Togelius. *Artificial Intelligence and Games*. Springer, 2018.

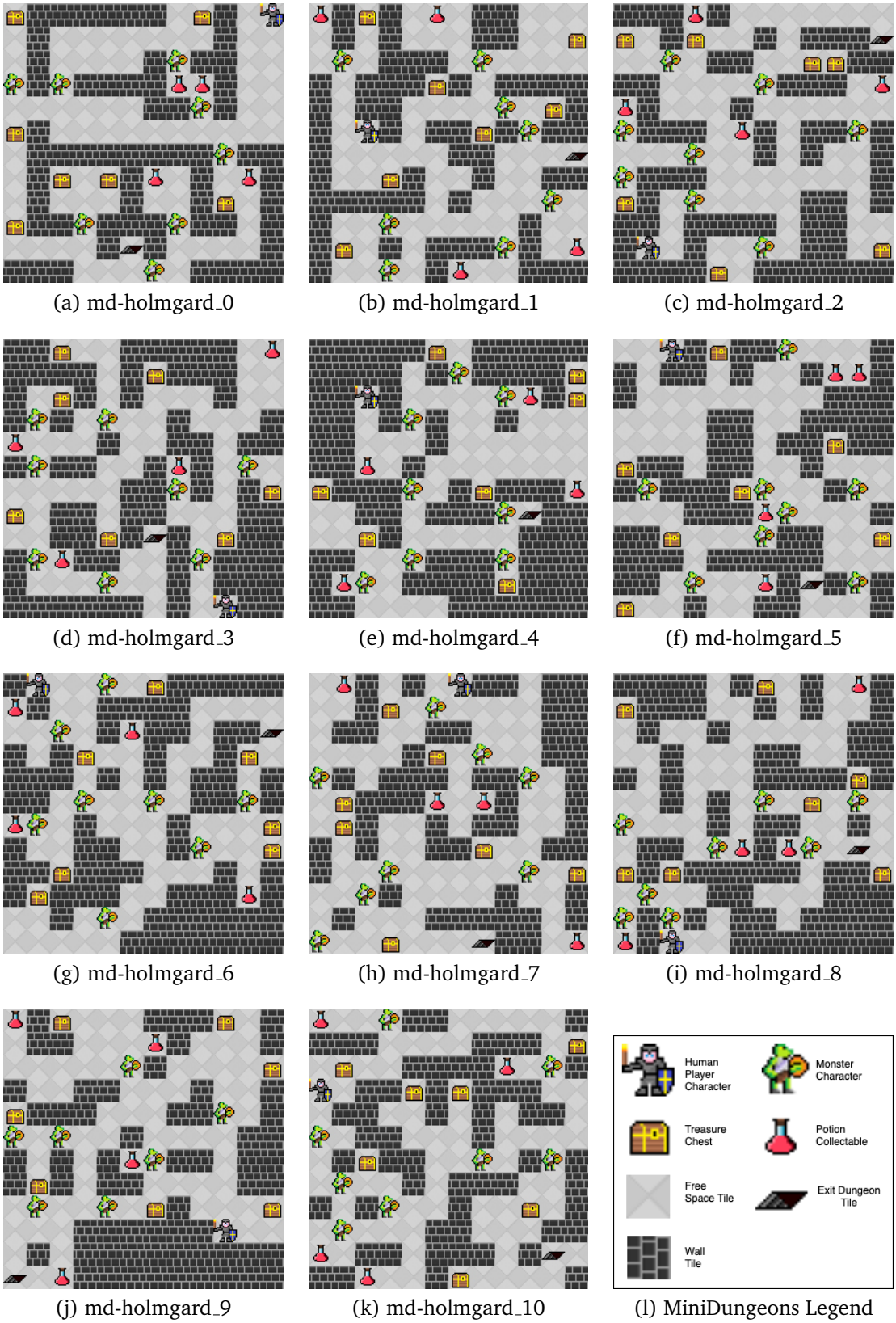
[Yannakakis *et al.* 2013] G. Yannakakis, P.H.M. Spronck, D. Loiacono, and E. Andre. *Player modeling*, pages 45–59. Number 6 in Dagstuhl Follow-Ups. Dagstuhl Publishing, 2013.

[Yannakakis 2008] Georgios N. Yannakakis. How to model and augment player satisfaction: a review. In *WOCCI*, 2008.

[Zhang and Verbrugge 2018] Mengxi Zhang and Clark Verbrugge. Modelling player understanding of non-player character paths. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 14(1), 2018.

Appendix A

Appendix: The MiniDungeons Game Domain Level Screenshots



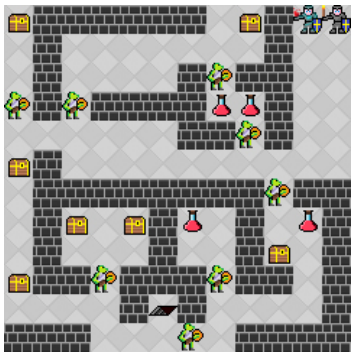
(a) md-holmgard_0 (b) md-holmgard_1 (c) md-holmgard_2

(d) md-holmgard_3 (e) md-holmgard_4 (f) md-holmgard_5

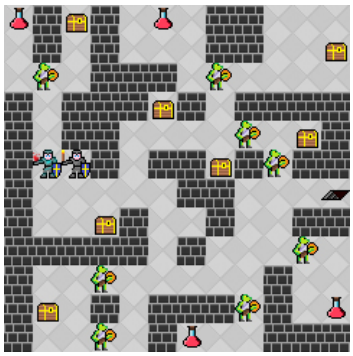
(g) md-holmgard_6 (h) md-holmgard_7 (i) md-holmgard_8

(j) md-holmgard_9 (k) md-holmgard_10 (l) MiniDungeons Legend

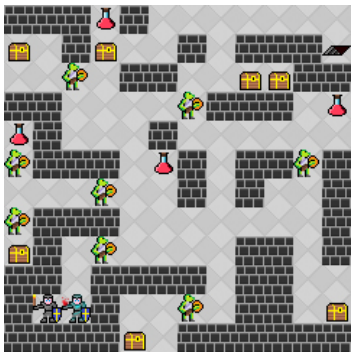
Figure A.1: Single Agent MiniDungeons Levels [Holmgård *et al.* 2014c][Holmgård *et al.* 2014b].



(a) md-collab-holmgard_0

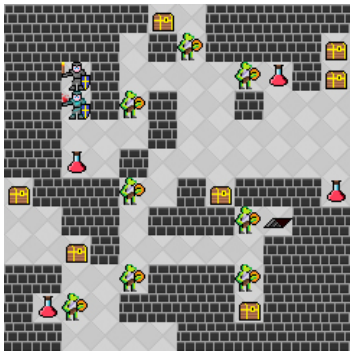


(b) md-collab-holmgard_1



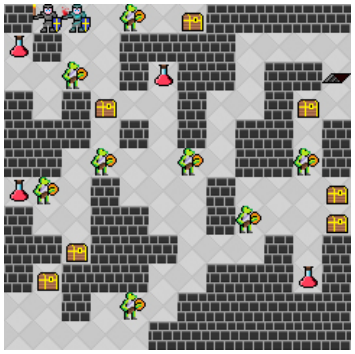
(c) md-collab-holmgard_2

(d) md-collab-holmgard_3

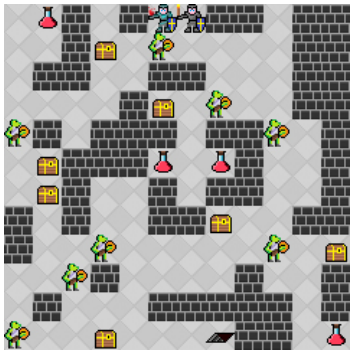


(e) md-collab-holmgard_4

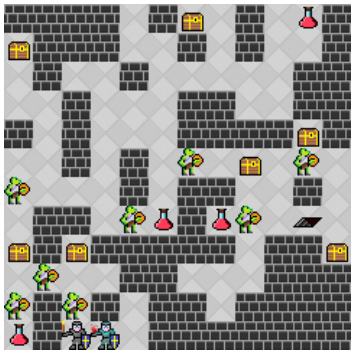
(f) md-collab-holmgard_5



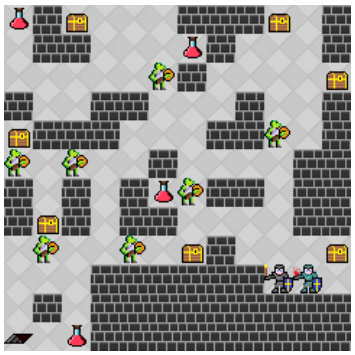
(g) md-collab-holmgard_6



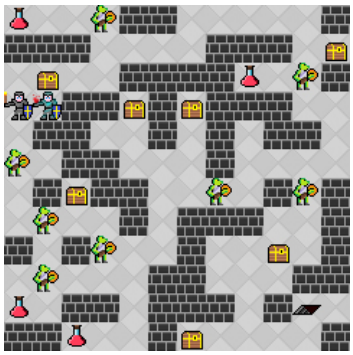
(h) md-collab-holmgard_7



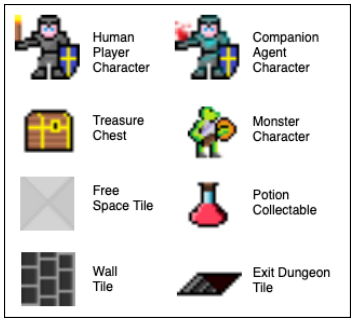
(i) md-collab-holmgard_8



(j) md-collab-holmgard_9



(k) md-collab-holmgard_10



(l) Collaborative MiniDungeons Legend

Figure A.2: Collaborative MiniDungeons Levels.

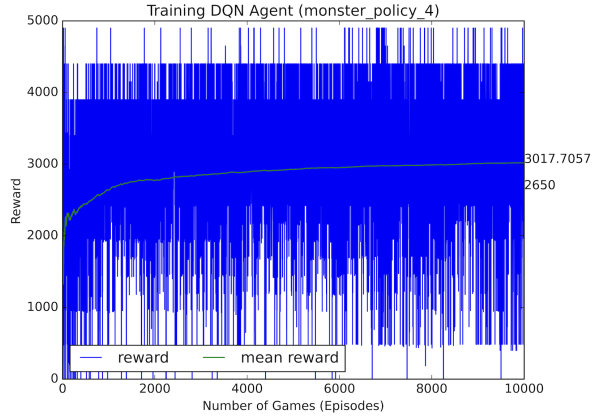
Appendix B

Appendix: Companion Agent Policy Creation

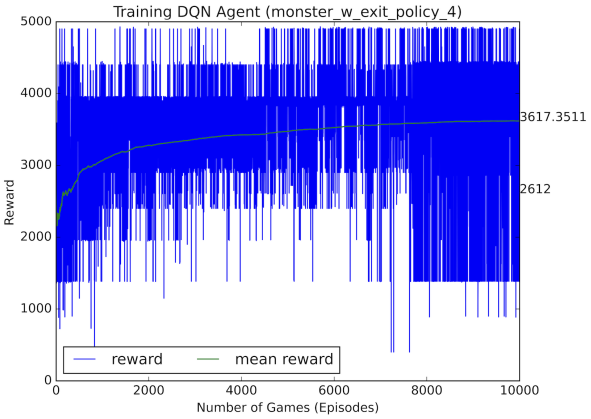
Algorithm 5 Human Proxy Playstyle and Companion Agent Policy Mapping Process

Require: L_{train}, Γ, Π, ϖ, $num\_episodes$, $episode\_length$

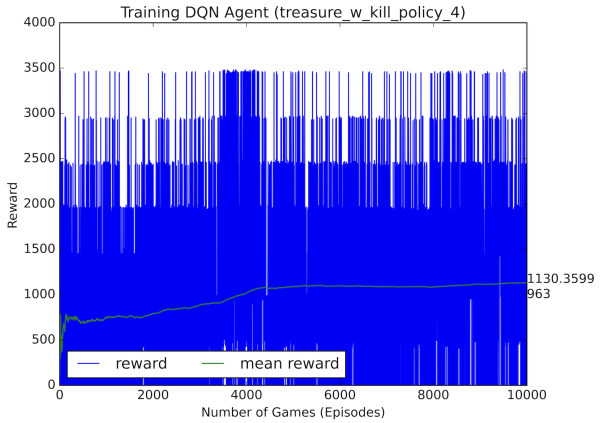
for level in L_{train} **do**
  for human_proxy_agent in Γ **do**
    human_proxy_agent_reward $\leftarrow$ get_reward_function(ϖ)
    for agent_policy in Π **do**
      for episode in $num\_episodes$ **do**
        env $\leftarrow$ initialise(level, human_proxy_agent_reward)
        done $=$ false
        for step in $episode\_length$ **do**
          proxy_action $\leftarrow$ human_proxy_agent.action(env)
          companion_action $\leftarrow$ agent_policy.action(env)
          env, reward, playlog_entry, done $\leftarrow$ env.step(proxy_action, companion_action)
          record_reward(human_proxy_agent, agent_policy, reward)
          if done is true **then**
            break
          end if
        end for
      end for
    end for
  end for
end for
reward_data $\leftarrow$ load_reward_dataset()
generate_playstyle_policy_mapping(reward_data)



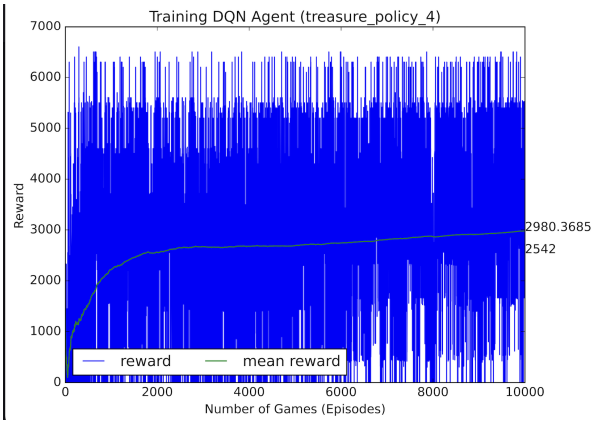
(a) monster_policy



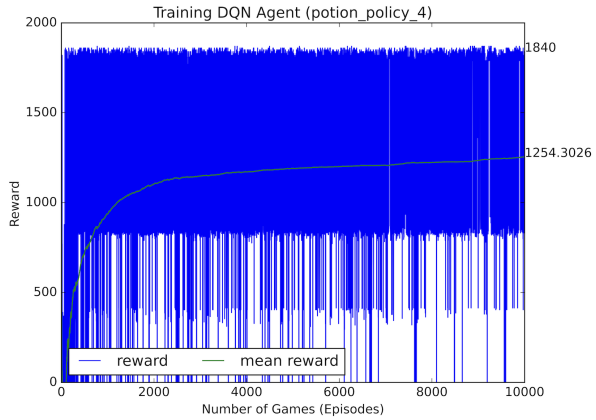
(b) monster_w_exit_policy



(c) treasure_w_kill_policy



(d) treasure_policy



(e) potion_policy

Figure B.1: Companion Agent Policy Training Graphs

Table B.1: Companion Agent Policy Reward Function Values

Policy	**Reward Values**
monster_w_exit_policy	{ “TURN”: 1, “EXIT”: 10, “KILL”: 500, “TREASURE”: 0, “POTION”: 250, “DEAD”: -100 }
monster_policy	{ “TURN”: 1, “EXIT”: -1000, “KILL”: 500, “TREASURE”: 0, “POTION”: 250, “DEAD”: -1000 }
treasure_w_kill_policy	{ “TURN”: 1, “EXIT”: -1000, “KILL”: 10, “TREASURE”: 500, “POTION”: 0, “DEAD”: -1000 }
treasure_policy	{ “TURN”: 1, “EXIT”: -1000, “KILL”: -100, “TREASURE”: 1000, “POTION”: 0, “DEAD”: -1000 }
potion_policy	{ “TURN”: 1, “EXIT”: -1000, “KILL”: -10, “TREASURE”: -10, “POTION”: 500, “DEAD”: -1000 }