Procedural Content Generation for video game levels with human advice

School of Computer Science & Applied Mathematics
University of the Witwatersrand

Nicholas Raal
793528

Supervised by Dr Steven James

July 20, 2023

Ethics Clearance Number: CSAM-2022-10

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

Abstract

Video gaming is an extremely popular form of entertainment around the world and new video game releases are constantly being showcased. One issue with the video gaming industry is that game developers require a large amount of time to develop new content. A research field that can help with this is procedural content generation (PCG) which allows for an infinite number of video game levels to be generated based on the parameters provided. Many of the methods found in literature can generate content reliably that adhere to quantifiable characteristics such as playability, solvability and difficulty. These methods do not however, take into account the aesthetics of the level which is the parameter that makes them more reasonable levels for human players. In order to address this issue, we propose a method of incorporating high level human advice into the PCG loop. The method uses pairwise comparisons as a way in which a score can be assigned to a level based on its aesthetics. Using the score along with a feature vector describing each level, an SVR model is trained that will allow for a score to be assigned to unseen video game levels. This predicted score is used as an additional fitness function of a multiobjective genetic algorithm (GA) and can be optimised as a standard fitness function would. We test the proposed method on two 2D platformer video games, *Maze* and *Super Mario Bros* (SMB), and our results show that the proposed method can successfully be used to generate levels with a bias towards the human preferred aesthetical features, whilst still adhering to standard video game characteristics such as solvability. We further investigate incorporating multiple inputs from a human at different stages of the PCG life cycle and find that it does improve the proposed method, but further testing is still required. The findings of this research is hopefully going to assist in using PCG in the video game space to create levels that are more aesthetically pleasing to a human player.

Declaration

I, Nicholas Oliver Raal, hereby declare the contents of this research report to be my own, unaided work. This research report is being submitted for the Degree of Master of Science in Computer Science at the University of the Witwatersrand, Johannesburg. This work has not been submitted to any other university, or for any other degree.

Acknowledgements

This research would not have been possible without the invaluable input and assistance provided by Dr. Steven James.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Procedural content generation (PCG) is an algorithmic process which allows for the automation of the design of video game components based on the requirements of the player or developer. These components range from textures, levels, enemies, characters and much more [Barriga 2018; Shu *et al.* 2021a]. PCG has many other applications, some of which are music and film [Barriga 2018]. A key element to note between game related PCG and PCG in other domains is that video game PCG has structural constraints that ensures the playability of the video game [Summerville *et al.* 2017]. The benefit of PCG for video games is that it allows for content to be generated without the need of a human game developer to manually design every element, increasing the efficiency of the game developers.

The field of PCG for video game levels has grown in popularity over recent years. PCG in video games has been used since the early 1980's in games such as Rogue and Elite. PCG was used in these games as a way to generate each dungeon, which allowed for replayability [Barriga 2018]. The most recent adoption of PCG in video games is from the developers of *No Man's Sky* which makes use of online PCG which enables it to generate an infinite number of planets in the expanding universe of the game [Barriga 2018].

PCG for level generation has had success when it comes to 2D video game levels [Barriga 2018; Shu *et al.* 2021a]. PCG can either be done in an offline or online manner depending on what is required from the game developers. The main difference being that online PCG is more dynamic and interactive but needs to be done in real-time in order to not negatively impact the game performance [Shu *et al.* 2021a]. Offline PCG on the other hand occurs during the games development stage.

An example of using online PCG is Shu *et al.* [2021a] who proposed a way in which player experience can be used to dynamically change the difficulty of the level, in real-time, based on the player's skill. Mawhorter and Mateas [2010] proposed a mixed method in which a level was made up of chunks of generated content and human de-

signed content which was done using offline PCG. Linden *et al.* [2013] used gameplay mechanics as the gameplay grammar that they used to generate action graphs which were used for the level layout. None of these approaches however tackle the aesthetics of the level being generated.

Using PCG in level design provides a very useful tool for game designers as it helps to shift their skills to other aspects of game design, whilst generating an unlimited number of levels. The issue at hand is that although PCG has been successful in the actual gameplay mechanics, such as the level being solvable and difficult, the actual aesthetics of the video game level does not yet match one that is reasonable to humans. Although the generated levels can be played by a human, they do not look appealing which can cause players to lose interest as well as removing the usefulness towards the feasibility of PCG for video game level generation. By improving on the aesthetics of video game levels such that they look more like those designed by a human game developer, the interest of a human player would be increased. This is an example of how PCG can be improved in such a way that it will become much more useful to the game design field than it is currently.

For the research conducted in this paper, 2D platformer video games, *Super Mario Bros* (SMB) and *Maze* are used in order test the proposed PCG method of bringing level aesthetics into the loop. Both of these games will be explored in order to give a more generalized view as both of these games look completely different, have different level sizes and follow different gameplay mechanics. Additionally, each of these video games have been used throughout literature making them ideal candidates [Khalifa *et al.* 2020; Yannakakis and Togelius 2011; Shu *et al.* 2021a; Beukman *et al.* 2022a].

Previous works on PCG in level design have not taken into account the human preference of the levels being generated. This is because the aesthetics of a video game level is not a standard metric that can be easily measured, as it is a human visual preference. But how would you go about assigning a metric to something like aesthetics? This research takes inspiration from the work of Raskar *et al.* [2015] where a ranking is assigned to Google Streetview images based on whether they look safe or not. By using a similar approach, we propose a way in which human advice can be incorporated into a PCG loop in order to generate levels that are more aesthetically pleasing to a human player. The approach will assign a score to a video game level based on the aesthetic preference of the human viewing the level. We will however require multiple levels so that the scores have structure such that higher scores resemble levels which are more aesthetically pleasing to the human. This will be done using a tournament structure where levels are considered contestants and the outcome is determined by pairwise comparisons. In doing so, levels with a better aesthetic preference will be assigned a score higher than those that are not aesthetically pleasing. By using these scores and feature vectors describing the level, a regression model can be developed that will be able to accurately assign a score to unseen generated video game levels. This score will then be used in conjunction with a fitness function of a genetic algorithm (GA) which

can be optimised as any other fitness function in order to generate new levels containing features the human finds aesthetically pleasing.

From the research conducted, we made use of TrueSkill, which is a probabilistic skill rating system developed for online gaming in order to match players with similar skills against one another. It was found that using TrueSkill to assign a score to a video game level by means of pairwise comparisons, and using the TrueSkill data along with the level feature data, a regression model can be trained and is able to successfully predict the TrueSkill scores for unseen levels. The regression model is used within a fitness function for a GA which can be optimised as any standard fitness function in order to bias the GA into generating levels which contain features that resemble a high TrueSkill score. Thus, it can be determined that the method used in this research can successfully be used to generate 2D video game levels that contain features which are aesthetically preferred by a human. Along with this, the results obtained show that characteristics of video games, such as solvability, are maintained in the newly generated levels. This finding is hopefully going to assist in the use of creating video game levels with PCG that are more aesthetically pleasing to a human player.

Previous work related to PCG and genetic algorithms as well as background information for each of the steps taken in the development of the implementation is discussed in Chapter 2. The goals of the research are presented and the implementation steps and design choices are described in Chapter 3. Chapter 4 provides the results and analysis for each of the experiments conducted. The conclusion to the findings of this paper are stated in Chapter 5.

Chapter 2

Background and Related Work

The field of PCG in video games can be broken down into multiple different elements, which range from rules, gameplay mechanics, puzzle solving, map design and level design to name a few. The research conducted in this paper focuses on the level generation in video games and as such the other elements will not be discussed in detail. The term level design in the context of this research refers to video game levels in 2D video games such as *Super Mario Bros* (SMB).

The next sections of the chapter will discuss what genetic algorithms are and how they have been used in a PCG space along with machine learning based approaches to PCG. This is followed by background information on each of the sections that make up the research methodology.

2.1 Genetic Algorithms

One of the most common approaches to PCG is to use genetic algorithms (GA) [Katoch *et al.* 2021; Togelius *et al.* 2011; Pedersen *et al.* 2009; Yannakakis and Togelius 2011]. A genetic algorithm is a search-based optimisation algorithm that took inspiration from the theory of natural evolution. By following natural evolution, the algorithm selects the fittest individuals from the population as the parents to future generations, thus following survival of the fittest. A typical genetic algorithm is described in the pseudocode shown in algorithm 1.

The algorithm begins with a set of individuals, commonly called a population. These individuals are made up of multiple genes which are encoded, mostly commonly using a binary encoding scheme, to form a chromosome, which is a string of genes. The purpose of the encoding is to improve the speed of the crossover and mutation steps. Other encoding scheme methods are hexadecimal, value-based and tree [Katoch *et al.* 2021]. A fitness function is used in order to calculate a fitness score for each of the chromosomes. This score is used as a metric to determine which chromosome will be

Algorithm 1 Standard Genetic Algorithm adapted from Katoch *et al.* [2021]

Require: $n \leftarrow$Population size
Require: $m \leftarrow$ Maximum number of iterations
Ensure: $Y_{bt} \leftarrow$Global best solution
  Randomly generate initial population of size n
  $t \leftarrow 0$
  Compute fitness function for each chromosome
  while $t \neq m$ **do**
    Select a pair of chromosomes based on fitness, c_1 and c_2
    Apply crossover operator with crossover probability C_p
    Apply mutation on offspring O with probability M_p to get O'
    Replace the old population with the new population
    $t \leftarrow t + 1$
  end while
  Return $Y_{bt} \leftarrow$best solution

selected for reproduction for the future generation. The next step is the selection phase where a pair of chromosomes need to be selected from the population, c_1 and c_2, based on their fitness score. Chromosomes with a high fitness score will have a higher chance to be selected, as they are the better picks for reproduction of the future generation. The convergence rate of the genetic algorithm is directly impacted by the selection step.

From the chromosome pair, a crossover operation is performed with a crossover probability of C_p in order to produce an offspring, O, which is added to the population. Typically a single point crossover is used, where the genes after the crossover point are swapped [Katoch *et al.* 2021]. The crossover point is chosen randomly within the genes of the chromosomes. The offspring is a chromosome that has a mixture of the genes from each parent until the crossover point is reached. As an example, consider individuals c_1 and c_2 as each parent and individuals c_3 and c_4 as their corresponding offspring. It is clear that c_1 and c_3 share the same four initial genes, and the same relationship occurs between c_2 and c_4. This is until the crossover point is reached, $|$, where the last four genes of the parents are switched in the offspring genes, thus the last four genes of c_1 and c_2 become the last four genes of c_4 and c_3 respectively. This relationship exists so that each offspring individual is made up of both parents genes.

$$c_1 = 0110|\mathbf{0011} \qquad c_3 = 0110|\mathbf{1110}$$
$$c_2 = 1100|\mathbf{1110} \qquad c_4 = 1100|\mathbf{0011}$$

The new offspring has a small probability, M_p, of its genes mutating which flips the genes in order to prevent premature convergence within the population. One of the simplest forms of mutation for a binary encoding scheme is by simply flipping some of the bits of the offspring [Katoch *et al.* 2021]. For example, consider a mutation occurring on the offspring c_3. Individual c_3' shows what a potential mutated c_3 could be if

the 4th, 5th and 6th bits were to be mutated. The mutated offspring, O', is then added to the population. Each of these steps are repeated until a new population has been generated [Katoch *et al.* 2021].

$$c_3' = 011\mathbf{100}10$$

A different crossover operation is two point crossover where there are two crossover points for which the parent individuals will swap their genes [Katoch *et al.* 2021]. As an example, consider individuals b_1 and b_2 where the crossover points are shown as $|$. When a crossover point is reached, the genes of the parents will swap. In two point crossover, there are two crossover points which means that the parents genes will swap twice, ultimately meaning that the middle segment in each parent is swapped in order to produce the offspring, b_3 and b_4.

$$b_1 = 011|000|11 \qquad b_3 = 011|011|11$$
$$b_2 = 110|011|10 \qquad b_4 = 110|000|10$$

2.1.1 Genetic Algorithms in PCG

Pedersen *et al.* [2009] proposed a way in which to personalize the level generation for *Infinite Super Mario Bros*, a modified version of the original game. The important features that must be added to the *Super Mario Bros* level are blocks, gaps and enemies in order for the level to be deemed complete. By modifying the pre-existing level generator, Pedersen *et al.* [2009] used four controllable parameters: number of gaps, average gap width, spatial diversity of the gaps and the number of direction switches, to generate the new levels. In addition to the controllable parameters, five statistical features were taken into account for the level generation: time, death, jump, item and kill. All of these parameters were combined into a single vector which is then converted into a level in a stochastic way. The vector is used as the input to an artificial neural network (ANN) which then maps to a potential emotional preference of a human player. The primary emotional preferences considered by Pedersen *et al.* [2009] were fun, challenge and frustration. The data used for the training of the network was collected over the internet using a questionnaire.

Sorenson and Pasquier [2010] proposed a different approach compared to most literature, in which the exact details of how a level should be generated are not required, rather the desired properties of the level to be generated are used. In order to achieve this, a genetic algorithm is used, more specifically, a Feasible-Infeasible Two-Population (FI-2Pop) genetic algorithm. The algorithm starts with an infeasible population, which consists of levels that do not adhere to the basic level requirements. These infeasible individuals follow the genetic algorithm to evolve individuals that satisfy all of the level

constraints, in which case they become the feasible population. The feasible population is evaluated using a fitness function which is ultimately used to generate the levels which satisfy the requirements of a playable level and are fun to the user. In order to determine whether or not the generated level is fun, Sorenson and Pasquier [2010] used a genetic method proposed in some of their earlier work where a fitness function is used to identify if a given level is fun or not. The approach was tested on both SMB and a Zelda like adventure game.

Yannakakis and Togelius [2011] proposed a framework for PCG that computationally models the user experience in order to provide the player with an optimised experience. The framework proposed is called experience-driven procedural content generation (EDPCG) which is built up of four key components: content representation, player experience modeling, content generator and content quality. In order to test the proposed framework, Yannakakis and Togelius [2011] used *Super Mario Bros* as a means to provide personalized level generation. The level itself was represented as a short parameter vector containing the number, size and location of gaps along with a boolean for the switching mechanic. Yannakakis and Togelius [2011] used data collected from hundreds of human players who ranked various levels in terms of fun, challenge, predictability, frustration, boredom and anxiety. The data was used for training a neural network which would predict the player states by using evolutionary preference learning. Although the framework was tested on *Super Mario Bros*, the concept behind the framework can be applied to various other 2D video games.

Togelius *et al.* [2010] proposed an approach to search-based PCG by making use of multiobjective evolution, whereas most literature only focus on single objective evolution, to generate complete maps in a video game. Togelius *et al.* [2010] chose the video game Starcraft to test their procedural content generation where entire maps were generated. A total of eight fitness functions were used in order to reflect the various game characteristics, such as playability, fairness, skill differentiation and interestingness. A state-of-the-art multiobjective evolutionary algorithm (MOEA) called SMS-EMOA (Hochstrate *et al.* [2007]) was used by Togelius *et al.* [2010] to ensure the individuals of the population remained within the region to ultimately achieve a sufficient representation of a Pareto front.

2.2 Machine Learning Approaches

Machine learning (ML) PCG has brought a new understanding to the PCG field as well as having the potential to expand the field in the near future. Summerville *et al.* [2017] surveyed methods of using machine learning in the PCG field. Machine learning improves autonomous generation compared to that of search-based methods as it removes the validation step that requires an evaluation function to validate the content since the inputs to the network can be specified for the content required [Summerville *et al.* 2017]. The downside to machine learning PCG is that machine learning requires

a lot of training data, which is often not the case for video games [Summerville *et al.* 2017]. The MLPCG approach by Summerville *et al.* [2017] focuses their content generation on functionality, whereas this research will focus on both functionality as well as the aesthetics of the generated content.

A downside to the ML approach to PCG is that its performance is dependent on training data. A reinforcement learning (RL) approach can perform similarly to ML, but does not have the same dependency on training data. One of the first frameworks for using reinforcement learning in a PCG setting was proposed by Khalifa *et al.* [2020]. The framework proposed by Khalifa *et al.* [2020] is shown in Figure 2.1. Due to RL being used, the process of PCG was seen as an iterative trask, rather than generating all of the content at once which is done in search-based methods. In order for this to be done, Khalifa *et al.* [2020] modeled content as a MDP. The iterative approach begins by having the level populated randomly with a set of tiles. At each state, S_t, the agent can change a single tile, A_t, within the level which then gets judged with respect to the goal of the level, producing a reward, r, for the agent. Khalifa *et al.* [2020] decided to generalize the framework so that it could be implemented in other games. This was achieved by splitting the framework into three parts: problem module, representation module and change percentage. The problem module stores all of the information about the video game level, examples include the reward function and goal. The representation module transforms the current level into an observation state that is viable for the agent. In the representation module, three different formulations of the actions were used: narrow, turtle and wide. For the narrow formulation, the agent is provided with the current state and a location where it is able to make a change. In turtle, the agent can move around at each time step and change the tiles along its path. For wide, at each time step the agent has complete control over the location and tile type [Khalifa *et al.* 2020]. The final module, change percentage, limits the number of changes the agent can make per episode which is used as a way to prevent the agent from making too many content changes. The change percentage works similarly to a discount factor in order to determine how greedy the agent should be. Khalifa *et al.* [2020] designed the agent with limited actions in order to force it to only be able to take a few actions in the environment. This was necessary as an optimal agent would always converge to a single optimal solution whereas that is not the desired outcome of the PCG level. Once the agents completed their training, they were used as generators for new levels, where they would alter randomly initialized levels for the given number of iterations. Khalifa *et al.* [2020] tested their framework on three simple video games: *Maze*, *The Legend of Zelda* and *Sokoban* where good results were found for *Maze*, however the agent was unable to effectively generate harder levels for Zelda and Sokoban.

Shu *et al.* [2021b] proposed a new framework that combines each of the frameworks proposed by Khalifa *et al.* [2020] and Yannakakis and Togelius [2011] into the ED(PCG)RL framework. The proposed framework allows for the generation of personalized content using reinforcement learning by making use of experience-driven reward functions. By building upon PCGRL, Shu *et al.* [2021b] have managed to generate endless online levels that adhere to aesthetical and functional properties. A latent vector is

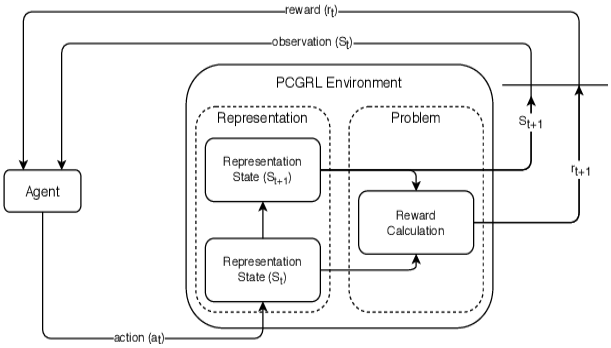


Figure 2.1: PCGRL system architecture framework for content generation proposed by Khalifa *et al.* [2020]

used to represent the state and action of the reinforcement learning agent, which allows the framework to be applied to any game that features levels which can be represented by segmentation and compression, for example 2D Atari games. A content quality component is used as a way to guarantee the content quality of the experience model. This quality is that which has been tested in gameplay simulations. The agent can then take an action corresponding to a generative act which alters the state of the content and then receives a reward via the experience model function. The state-action and experience representations are traversed iteratively in order to find a design policy that will optimise the experience model. Shu *et al.* [2021b] tested their framework on *Super Mario Bros* where the framework can be broken down into three components: repairer and generator of non-faulty segments, artificial SMB player to test segment playability and reinforcement learning agent that plays the *Super Mario Bros* endless-platform generation game. For the generator, MarioGAN proposed by Volz *et al.* [2018], is used along with CNet-assisted Evolutionary Repairer to successfully generate the level segments. The approach differs from PCGRL as the latent vector of MarioGAN is used to represent the agent's states and actions for each level segment which is faster than the approach by Khalifa *et al.* [2020].

Jennings-Teats *et al.* [2010] proposed an online level generator which changed its difficulty based on the players skill level in real-time. The game of choice was Polymorph, a 2D platformer game which adjusts its levels during playtime using dynamic difficulty adjustment (DDA). This implies that segments of the level need to be generated as the player traverses further and further, however the difficulty of each segment is changed to be appropriate to the skill level of the player. The reasoning behind creating a DDA is to attempt to avoid the player being frustrated or bored with the game. This approach has elements of search-based algorithms, however it's described as a machine learning-based solution to dynamic video game level generation.

By using quality diversity, Gravina *et al.* [2019] proposed a method of PCG by using a subset of the work by Togelius *et al.* [2011]. The reasoning as to why Gravina *et al.* [2019] used a quality diversity approach to PCG is that in many PCG problems, there is a tradeoff between the quality and the diversity of the generated content. A quality-

diversity (QD) algorithm is one in which the quality and diversity of the solution is simultaneously maintained by rewarding divergence and using constraints to control the solution's quality. Gravina *et al.* [2019] ran experiments with multiple different use cases and QD algorithms to generate a range of different video game content. Only the approach used to generate level content will be focused on in this paper. A MAP-Elites algorithm was used by Gravina *et al.* [2019] to generate levels in *Super Mario Bros*. A feature map of 256 cells was generated, and the algorithm could generate levels in an average of 100 cells where levels with and without game mechanics were generated.

One aspect of PCG which has not been investigated is the lack of structure to the content that is pleasing to humans. This is because assigning a score to a video game level based off of a human preference and feeding it back into PCG is a challenging task. Similar methods have been used in a non-PCG setting. One such method which is used for predicting the safety of streets based off of Google Streetview images was proposed by Raskar *et al.* [2015], where a TrueSkill score is assigned to each image based on the results of 208738 pairwise comparisons. These scores were then used along with detected features from the Streetview images to train a v-SVR model in order to predict the safety of unseen streets.

2.3 Research Framework

This section provides an overview and background information relating to each of the steps in the proposed implementation.

2.3.1 TrueSkill

In order to provide a numeric value to the aesthetics of a video game level, imagine that each video game level is a contestant in a competitive tournament. Within this tournament, the winner is decided based on a human's visual preference when comparing one level to another. A way in which to reliably provide a numerical value to each contestant so that a ranking for each contestant can be determined is to use TrueSkill.

Trueskill is a probabilistic skill rating system which was developed for online gaming which assumes players in a game have respective skill levels, w_1 and w_2, and that game outcomes can be predicted. This is achieved by using the performance difference between skills which are subject to Gaussian noise effects [Herbrich *et al.* 2007; Minka *et al.* 2018].

TrueSkill is widely used by Xbox Live for competitive games, such as Halo and Forza Motorsport, as it allows for players with similar skill levels to be matched against one another for a level playing field. The rating system uses two numbers in order to calculate a players skill: average skill of a player, μ, and the degree of uncertainty in a

players skill, σ. By using a Guassian distribution a player's skill can be calculated to be between $\mu \pm 3\sigma$ with a 95% confidence [Herbrich *et al.* 2007; Minka *et al.* 2018]. The Guassian distribution used starts from $\mathcal{N}(25, \frac{25}{3}^2)$, which sets a new player to have a mean skill of $\mu = 25$ and an uncertainty of $\sigma = 8.333$ [Herbrich *et al.* 2007; Minka *et al.* 2018]. Figure 2.2 shows how the belief of the TrueSkill rating system is drawn.

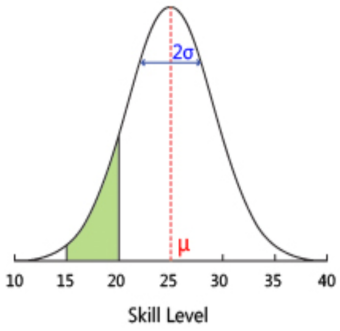


Figure 2.2: TrueSkill Gaussian belief curve [Herbrich *et al.* 2007; Minka *et al.* 2018]

In situations where the uncertainty is high, the system does not know exactly what the player skill should be, whilst a low uncertainty means that the ranking system strongly believes the player is close to the average. By using the degree of uncertainty, the system can make large changes to the skill estimates at early stages of new players calibration matches, however after multiple games only small changes in skill can be made. This allows the rankings system to identify the skill levels of players from a small number of games. Depending on the game mode and number of players, the ranking system can take long or short to determine a player's skill. By expanding on TrueSkill, TrueSkill 2 can determine the skill of a new player much faster, with an improved overall accuracy from 52% to 68%, however, TrueSkill 2 is not used in the research being conducted [Minka *et al.* 2018]. TrueSkill 2 was not used in this research as the *trueskill* Python library does not currently support TrueSkill 2; therefore implementing TrueSkill 2 from scratch would have been beyond the scope of this research.

A question that comes to mind is why use TrueSkill over other rating systems available? TrueSkill was developed for online gaming, however it is directly applicable to the ranking of images in a competitive type system, such as pairwise comparisons, which create one-versus-one match ups [Herbrich *et al.* 2007; Minka *et al.* 2018; Raskar *et al.* 2015; Burke 2016]. A 2D video game level can easily be transformed into an image, making TrueSkill directly applicable to generated 2D video game content.

TrueSkill when used in a two-player contest uses the equations shown in equation 2.1 in order to update the scores and uncertainties of player $p1$ and player $p2$, given the assumption that player $p1$ is the winner. In equation 2.1, $t = \mu_{p1} - \mu_{p2}$ and $c^2 = 2\beta^2 + \sigma_{p1}^2 + \sigma_{p2}^2$. The probability of a draw is given by the parameter ε, $v(t, \varepsilon)$ is a weighting factor for the players skill level and $w(t, \varepsilon)$ is a weighting factor for the

uncertainty [Liu *et al.* 2013].

$$
\begin{aligned}
\mu_{p1} &= \mu_{p1} + \frac{\sigma_{p1}^2}{c} \cdot v\left(\frac{t}{c}, \frac{\varepsilon}{c}\right) \\
\mu_{p2} &= \mu_{p2} - \frac{\sigma_{p2}^2}{c} \cdot v\left(\frac{t}{c}, \frac{\varepsilon}{c}\right) \\
\sigma_{p1}^2 &= \sigma_{p1}^2 \cdot \left[1 - \frac{\sigma_{p1}^2}{c^2} \cdot w\left(\frac{t}{c}, \frac{\varepsilon}{c}\right)\right] \\
\sigma_{p2}^2 &= \sigma_{p2}^2 \cdot \left[1 - \frac{\sigma_{p2}^2}{c^2} \cdot w\left(\frac{t}{c}, \frac{\varepsilon}{c}\right)\right]
\end{aligned}
\tag{2.1}
$$

2.3.2 Learning a Model

A model needs to be learnt that can map video game levels to a predicted aesthetic score. One method of doing this is to first compute image features from the levels and using the feature data to train a predictor model. 2D video game levels can be portrayed as 2D images which allows for computer vision image feature detection techniques to be used. Variations of the feature detection techniques, listed below, have been used in past research for scene recognition and passing feature data into a predictor. Although past research made use of natural images, the techniques should perform similarly on video game levels which makes them ideal techniques to potentially use for the research being conducted [Raskar *et al.* 2015; Xiao *et al.* 2010]:

- Histogram of local binary patterns (LBP)
- Speeded-Up Robust Features (SURF)
- Scale-invariant feature transform (SIFT)
- Histogram of Oriented Gradients (HOG)

LBP compares the current pixels value to each of is neighbouring pixels. If the neighbouring pixel's value is greater than or equal to that of the current pixel, assign the neighbouring pixel a 1. For any neighbouring pixel values less than the current pixel value, a 0 is assigned. A histogram is then created for the frequency of each value and used to form a feature vector [Ojala *et al.* 2002]. SURF is performed by approximating second order Gaussian derivations by using integral images that are created from a set of 2D box filters. This step is followed by a scale invariant blob detector which is uses to find the keypoints in the image, from which the image descriptors are described by the distribution of Haar wavelet responses near each keypoint [Mai Thanh Nhat Truong 2016]. A popular feature detection technique is SIFT in which keypoints are detected by finding areas with large gradients in more than one direction, based off of the Difference-of-Gaussian (DoG) operator. For each keypoint detected, a feature vector is created by using a histogram of the gradient orientations in a local area around the keypoint [Mai Thanh Nhat Truong 2016]. HOG divides the image into small cells and computes the magnitude and gradient for each pixel in the cell. This information

is then aggregated into a histogram of oriented gradients bin for each cell [Bertozzi *et al.* 2007].

A potential second method for feature extraction from the input video game levels is to use an neural networks in the form of an auto-encoder. This consists of encoding the input image into a vector and then decoding the vector back into the original image whilst minimizing the error produced when reconstructing the original input image. An issue with this method is that it requires a large amount of data in order to train the encoder and focuses on quantity of data in the video game level rather than important data in the video game level. Video game levels have a relatively low amount of data, but a high amount of key features which directly impacts the aesthetics of the level. Due to how video game levels are comprised and the issue of requiring large amounts of training data for auto-encoders, the approach of extracting feature data using more traditional techniques is the preferred method in this research.

Once the various features for the images have been computed, a regression model can be trained which maps the features to the TrueSkill score. This model can then be used to predict the TrueSkill score for any new unseen video game level images as it takes an image feature vector as an input, and outputs a predicted TrueSkill score. The predicted score can then be used in the fitness function of a GA.

In supervised machine learning, there are a few different types of regression models each of which have their own advantages and disadvantages. This research decided to use support vector regression (SVR) due to its advantages of being able to generalize well for smaller datasets, such as the feature vectors, whilst providing a high accuracy in its prediction capabilities and providing a way in which the model variables can easily be configured for each application. Additionally it allows for the maximum number of datapoints to be used within the hyperplane which ensures the majority of the small dataset is accounted for which allows for subtle patterns within the dataset to be identified [Najwa Mohd Rizal *et al.* 2022].

SVR is an extension to the popular supervised machine learning tool support vector machine (SVM), first introduced by Vapnik [1995]. SVM is commonly used in classification based solutions whilst SVR is more commonly used in regression based solutions [Smola and Schölkopf 2004; Hsu *et al.* 2003]. In order to predict a TrueSkill score for an unseen level, a regression model can be used where the unseen level feature data is taken in and used as the basis for predicting a TrueSkill score.

There are several parameters used in SVR which can be optimised in order to improve the accuracy of the model, some of which are discussed in detail. These parameters are:

- C: the regularization parameter

- γ: the kernel coefficient
- Kernel: the kernel type

The regularization parameter, C, controls the tradeoff between minimizing the error between the training and testing data, thus determining how generalized the model is when it comes to unseen data [Smola and Schölkopf 2004; Hsu *et al.* 2003]. As this research requires accurate prediction on unseen feature data, selecting the correct C can make or break the model. One method of ensuring C is tuned correctly is to use cross-validation to prevent overfitting, or a grid search [Smola and Schölkopf 2004; Hsu *et al.* 2003]. The γ parameter directly affects the kernel being used and primarily has two options, the first being *scale* which uses a value of $\frac{1}{(num\_features * X.var())}$ and the second being *auto* with a value of $\frac{1}{num\_features}$ where $num\_features$ is the number of input data features and $X.var()$ is the variance of the input data features. Similarly to C, cross validation and grid search are commonly used to tune γ [Smola and Schölkopf 2004; Hsu *et al.* 2003]. The last parameter that is optimised is the kernel, which in the case of this research, alternates between *linear* and *radial basis function* (*RBF*). An *RBF* kernel is a complex polynomial kernel which maps non-linear separable data samples into a higher dimension, which means that it is the preferred kernel when the data has a non-linear relationship [Hsu *et al.* 2003]. The *linear* kernel is preferred when the relationship between the data is linear. Often at times the relationship of the data is linear, however the *RBF* performs better and is thus chosen over the *linear* kernel. This is due to the *linear* kernel being a special case of the *RBF* kernel [Hsu *et al.* 2003].

Raskar *et al.* [2015] made use of an improved SVR by Scholkopf *et al.* [2000] called v-SVR. The benefit of v-SVR over a normal SVR is that it provides a variable, v, which allows the number of predictions to be changed depending on the value of v and ε, essentially giving some slack to the regression model [Raskar *et al.* 2015; Scholkopf *et al.* 2000]. In SVR a regression function, $f(\boldsymbol{x})$, can be calculated using equation 2.2 where $\boldsymbol{x}$ is the input feature vector and $\boldsymbol{y}$ are the corresponding labels. In order to estimate functions using equation 2.2 the following is done. For each $\boldsymbol{x}_i$ an error of ε is allowed, where everything above ε uses slack variables $\xi_i^{(*)}$ to be captured. These slack variables are penalized in an objective function using a regularization constant, C, given a priori. Then, by minimizing equation 2.3 the accuracy of the approximation can be determined [Scholkopf *et al.* 2000].

$$f(\boldsymbol{x}) = (\boldsymbol{w} \cdot \boldsymbol{x}) + b \qquad \boldsymbol{w}, \boldsymbol{x} \in \boldsymbol{R}^N, b \in \boldsymbol{R} \tag{2.2}$$

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\tau}(\boldsymbol{w}, \boldsymbol{\xi}^{(*)}, \varepsilon) = \frac{1}{2}||\boldsymbol{w}||^2 + C \cdot \left(v\varepsilon + \frac{1}{l}\sum_{i=1}^{l}(\xi_i + \xi_i^*) \right) \\ \text{subject to} \quad & ((\boldsymbol{w} \cdot \boldsymbol{x}_i) + b - y_i \leq \varepsilon + \xi_i \\ & y_i - ((\boldsymbol{w} \cdot \boldsymbol{x}_i) + b) \leq \varepsilon + \xi_i^* \\ & \xi_i^{(*)} \geq 0, \varepsilon \geq 0 \end{aligned} \tag{2.3}$$

It was proved by Chang and Lin [2002] that if you were to use epsilon supported vector regression (ϵ-SVR) with the parameters ($\frac{C}{l}$, ϵ), it will have the same solution as v-SVR with the parameters of (C, v), where l is the domain of the input data, thus showing that using ϵ-SVR for this research with the correct parameters is a viable method and equation 2.2 can also be used to represent ϵ-SVR.

Various approaches from literature have been discussed on PCG in video games, such as genetic algorithms and machine learning based methods. Although plenty of work has been done in the field, PCG is still not widely used. This chapter discussed a non-PCG based method which can be translated into a PCG context for assigning a score to an unseen video game level based off of its aesthetics. The theory behind TrueSkill and the way in with an SVR model can be learnt from image feature data has been explored in this chapter. In the next chapter the goals of the research and the implementation that will be followed in order to solve the lack of aesthetically pleasing PCG based video game levels using a GA is addressed.

Chapter 3

Research Approach

Using the background information discussed in Chapter 2, it is evident that although many of the methods can procedurally generate video game level content that adheres to a playable level, none of these methods take into account the level design aesthetics. This issue arises due to PCG methods focusing their fitness functions on characteristics such as playability, completeness and level difficulty. Each of these characteristics are quantifiable; however it is far harder to quantify what is meant by an aesthetically pleasing video game level. Thus this research is aimed at providing high level human-in-the-loop advice to GA based PCG. It is important that the advice provided by the human is high level, in order to allow anyone to provide insight, whilst being infrequent, such that the human is not constantly required to provide the advice.

This chapter sets out the goals to be achieved along with the experimental design and choices made for the research conducted.

3.1 Goals

The purpose of this research is to determine whether taking in human advice, converting it to a fitness function and then applying it to a genetic algorithm can result in more aesthetically pleasing procedurally generated video game levels, based on the specified human input, whilst still adhering to video game level characteristics.

The research being conducted proposes a way in which aesthetically pleasing levels can be quantified. To accomplish this, the research proposes using pairwise comparisons which will be shown to humans or virtual humans to allow them to select one of the two levels they find more aesthetically pleasing. Virtual humans in this research refer to code which will perform the role a human typically would in the pairwise comparisons. By using code, a human will not be required which allows for large scale experiments to be run quickly. The question then is how will pairwise comparisons transfer into a quantifiable measurement? The goal of this research is to use the

TrueSkill ranking system to systematically rank the levels based on a one-versus-one competition, being the pairwise comparisons. Thus the level the human prefers will be considered the winner of the one-versus-one competition and will be given a higher TrueSkill score compared to a level which is not found aesthetically pleasing. Following this, a predictor will be trained on the TrueSkill score data along with the corresponding level feature vector to predict TrueSkill scores for unseen levels. By using the predictor as a fitness function to the GA, the fitness can be optimised along with any additional fitness functions, such as solvability, in order to create a bias towards generating levels that a human would aesthetically prefer whilst still satisfying standard video game level metrics. This fitness function is referred to as the human fitness in this research.

The reasoning to include computational metrics, such as solvability, along with the human fitness is that this research did not want to merely create an interactive genetic algorithm (IGA) that constantly requires human input. But rather a GA that is built up of both a computation and interactive GA which requires a small amount of human input and show how they can work together.

An additional question which this research aims to answer is whether or not using human comparisons additional times during the PCG life cycle will improve the aesthetics of the levels being created.

3.2 Hypothesis

Using human advice for pairwise comparisons will provide a way in which to systematically rank unseen video game levels that allows for a optimisation objective which can be integrated into a fitness function, which will improve the aesthetics of the video game level whilst still adhering to set performance characteristics.

3.3 Experimental Design

In order for the presented hypothesis to be investigated and proved, experiments needed to be conducted. This section summarizes each aspect of the experimental framework used in order to conduct the necessary experiments and generate the results discussed in Chapter 4.

To reduce the risk of using human input and evaluation, a proof of concept will be developed prior to using real people to show that the method of developing levels based on human preference does in fact work. This will be done by constructing a virtual human and specifying a preference in level features. The pairwise comparison step will occur, but it will be completed by code using the binary level data, rather than a human selecting a level manually. This allows for 'human' advice to be added to the

PCG loop, without actually using a human.

Figure 3.1 provides a high level overview of the experimental framework and how the PCG loop will incorporate an SVR model to predict a human fitness value for unseen video game levels. For the very first iteration of the GA, the SVR model object does not yet exist, thus it is important that a random initial population of levels is generated which will aid in the creation of the model. The initial dataset will be randomly generated using a genetic algorithm; this way a large amount of level data can be generated without the need of using external resources. Pairwise comparisons to find preferred aesthetic features will then be done on this initial set of data, in the form of TrueSkill scores, along with feature vectors to describe each level. This data is preprocessed and used to train an SVR model which is passed back into the human fitness function of the GA. Once the model is passed back into the human fitness function, it is used to predict a TrueSkill score for each of the subsequent levels generated at each iteration of the GA.

An assumption being made is that the SVR model should improve each time pairwise comparisons are done as the regression model will be refitted to correctly distinguish more reasonable human levels.

3.3.1 Genetic Algorithm

The goal of the genetic algorithm is to optimise the human fitness function, whilst still adhering to the solvability fitness function. This means that although human fitness is the focus, the solvability fitness is just as important as newly generated levels should be both solvable and prefer the chosen human components.

The genetic algorithm implemented in this research is adopted from a general genetic algorithm developed by and used in Beukman *et al.* [2022a]. The implemented GA takes in a game level object and a fitness function. The GA aims to maximize the fitness function and selects the best performing individual, based on the fitness function, in order to select the best performing individual to breed with to form the next population. In the GA breeding step, the best performing individual is kept whilst the rest are generated using a simple two point crossover with the parents and a mutation probability.

The pseudocode for the steps required for incorporating human fitness into the GA to allow for PCG is shown in algorithm 2. Each of the steps taken are discussed in detail as well as design choices made.

One of the conditions of the research hypothesis is for the generated level to adhere to performance characteristics, whilst improving in its aesthetics. In order for this to occur, multiple fitness functions are required to be input into the GA. In order to com-

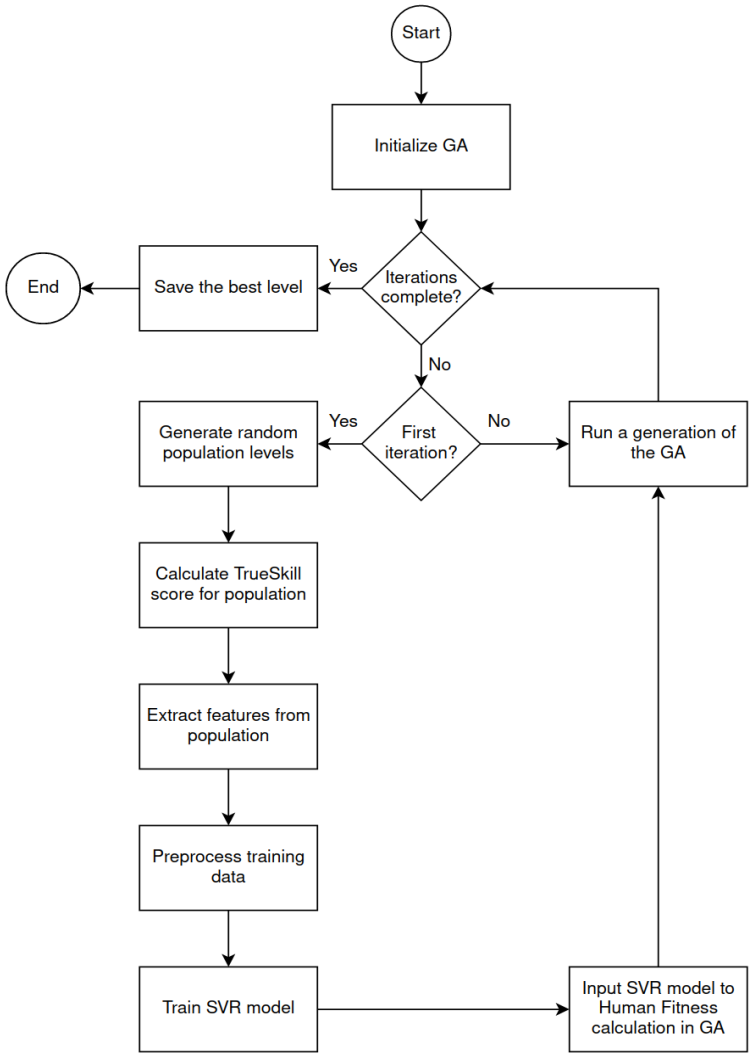


Figure 3.1: Flow diagram showing a high level overview of the experimental structure for the PCG process.

bine multiple fitness functions together, a weighted sum is combined with the fitness functions to create an overall fitness which is passed back into the GA. The method of using a weighted sum has been found to be comparable to multi-objective optimisation techniques [Beukman *et al.* 2022a; Gomes *et al.* 2015]. This overall fitness is essential for the improvement of the GA, but it is important to note that the individual fitnesses are also returned in order to record the performance of the GA during its life cycle.

In order for the human fitness function to be created, a set of initial population levels is required. This is because an SVR model cannot be developed without initial data. As such, the GA creates a completely random population, of a size specified by the user. Pairwise comparisons are applied to each level in the random population in order to assign TrueSkill scores to the data. The feature vectors for each population level are extracted and passed into the SVR object, along with the TrueSkill scores as

Algorithm 2 Procedural Content Generation with Human Fitness and Solvability

Require: $pop_size \leftarrow$Population size
Require: $ga\_iter \leftarrow$ Number of GA iterations
$hf\_weight \leftarrow$ Human fitness weight
$solv\_weight \leftarrow$ Solvability fitness weight
Initialize *CombinedFitness()* $\leftarrow$ Fitness functions and weights
Initialize *GeneralGAPCG()* object $\leftarrow$ Game level and *CombinedFitness()*
$model \leftarrow$ None
for i `in range(`ga_iter`)` **do**
    if $i == 0$ **then**
        Generate a random population of size pop_size
        $get\_trueskill\_scores() \leftarrow$ Calculate TrueSkill scores for each population level
        $generate\_features() \leftarrow$ Extract features from each population level
        $\boldsymbol{X}, \boldsymbol{y} \leftarrow preprocess\_data()$
        $model \leftarrow create\_SVR\_model(\boldsymbol{X}, \boldsymbol{y})$
        Update human fitness with optimised SVR $model$ object
    end if
    Perform an iteration of the GA
    Record best level fitness function values
end for
$best\_level \leftarrow$ Save best level

the training data for an SVR model.

Once a model has successfully been created, it is passed into the human fitness calculation to be used on unseen levels generated by the GA. The GA can then be run with the inclusion of a human fitness function for *N* number of iterations, *N* being specified by the user, and the best performing level of the GA is output. For each *N*, the GA runs for 100 iterations and selects the best performing individual to breed the next population. This individual is recorded and used to track the performance at each iteration of the GA.

3.3.2 Pairwise Comparisons

A pairwise comparison is when two images of levels are compared to one another and one of the images is preferred over the other in an aesthetic aspect. In order for the pairwise comparisons to be done by humans, we would treat the comparisons as a one-versus-one game. The game will show a pair of images to each human who are simply required to select either the left image or the right image, depending on their preference. The image that is selected will be deemed the winner, and the one that was not selected will be the loser. Based on many of these games being played with multiple different pairs of images, a ranking will be assigned to each image so that there is an overall outline of which images looked more aesthetically pleasing and which did not. Multiple games are played in order to provide enough data to train an acceptable

model. This approach is similar to how ranking is determined in a one-versus-one tournament structure.

Using an initial random population, humans or virtual humans, will be required to perform the pairwise comparisons on the population before a predictor can be developed. When using real humans, ethical clearance is required however no personal information of the humans will be recorded, only the outcome of the pairwise comparisons. The ranking system behind the pairwise comparisons will be TrueSkill, which is discussed in detail in Section 3.3.3. From the results of the pairwise comparisons, a regression model will be developed to predict the TrueSkill scores for unseen levels created by the GA.

For future work, where real humans are used, a way in which to perform the pairwise comparisons is to use a simple website which will allow the human to select the left or right image and then moves onto a new set of images until all of the pairwise comparisons have been complete.

3.3.3 TrueSkill

TrueSkill is a probabilistic skill based rating system which is used to assign a numerical value to each of the generated levels, be it in the initial GA population or during the GA generation process. Each level in the population is modeled by $N(\mu, \sigma^2)$ [Herbrich *et al.* 2007]. The calculation of the scoring is determined based on the features in the level being favoured by the human in a one-versus-one competition setting. Although TrueSkill does cater for a draw in a one-versus-one setting, a design choice was made to not implement the ability for two levels to draw. This design choice was chosen as having levels drawn would not provide any benefit to the outcome of the overall TrueSkills of each level. For example, in the virtual human case, if two levels contain the same number of black border pixels, the first level will be deemed the winner. Due to the nature of the round robin tournament, levels with identical virtual human features will be very similar in TrueSkill scores proving a draw condition is not required.

The TrueSkill score for the winning level is increased by its degree of uncertainty, σ, whilst the TrueSkill score of the losing level is decreased by its σ. The degree of uncertainty is a unique value for each level, which allows for large changes in TrueSkill scores for levels with a low number of games, and much lower changes the closer the level gets to its perceived skill. As per the documentation provided by Microsoft, in order for a stable TrueSkill score to occur for a level, a minimum of twelve contests is required [Herbrich *et al.* 2007]. However, in the case of a round robin tournament that contains more that eight contestants, the use case in this research, a minimum of only three games is required for a stable ranking [Herbrich *et al.* 2007].

As previously discussed, a round robin tournament style was chosen for the virtual human TrueSkill calculations. A round robin tournament is a tournament in which every competitor players one another once. This design choice was made for the virtual human implementation for two reasons. The first being that as it is virtual, a round robin competition can complete quickly, which leads into the second reason; of allowing for enough games to be played in order to ensure the degree of uncertainty is minimized for each competitor.

Once the round robin tournament has completed, the TrueSkill scores are stored so that they can be passed into the data preprocessing step before the SVR is trained. Due to the nature of the TrueSkill algorithm in estimating level skill, rather than level rank [Herbrich *et al.* 2007], a negative TrueSkill score can occur for a badly performing level. This negative score can transfer to the SVR model which in turn produces negative probabilities which cannot be used in the GA. As a way to ensure that the GA worked correctly for negative predicted scores, clamping is used which sets any negative predicted TrueSkill to 0. An alternative approach to clamping would be to scale the TrueSkill scores such that they could never go below 0. An issue with scaling the TrueSkill at this step is that it creates an additional tuning factor which would preferably be avoided to reduce the complexity in the code. As such, the weight of the human fitness can be altered easily which is used to scale the TrueSkill.

3.3.4 Feature Detection and Extraction

The previous section discusses how this research assigns a TrueSkill score to a level by comparing its aesthetics to other levels. The question that needs to be answered is: How will a TrueSkill score be predicted for an unseen level if there is nothing to compare the unseen level to? While neural networks seem like the natural choice to answer this question, we do not have a lot of data thus we will go for the method of extracting feature data from population levels and use that to train a regression model in which to predict the TrueSkill score.

There is a wide range of feature extraction techniques that have been researched extensively and used on a magnitude of different images and image types, but not many have been used on 2D video game levels. As such, there is currently no known research conducted which investigates the best feature extraction techniques for video game levels. Thus, this research implements five well known feature extraction techniques in order to find the best performing technique to extract feature descriptors from 2D video game levels and pass them into an SVR model. Some of the techniques chosen have been used in similar research focused on scene recognition, where they are used to extract features from natural images and then passed into an SVR model [Raskar *et al.* 2015; Xiao *et al.* 2010]. This step in the process is essential as the better the extracted features from the levels, the better the SVR model will perform on unseen data.

The five techniques implemented are:

- Scale Invariant Feature Transform (SIFT)
- Histogram of Oriented Gradients (HoG)
- Local Binary Patterns (LBP)
- Oriented FAST and Rotated BRIEF (ORB)
- Binary Robust Independent Elementary Features (BRIEF)

Each of these methods extract feature keypoints and descriptors from an input image and output them into a feature vector. The size of these feature vectors vary based on the method used; more powerful methods, such as SIFT and HoG, can detect 10000 features and more. With so many features and varying feature vector sizes a design choice was made to standardize the feature vectors and scale them down to a more manageable level whilst still providing enough information to not negatively affect the SVR model. This standardization was done by implementing a bag of visual words (bovw) approach to the feature extraction techniques which creates a frequency histogram for the features.

The bovw implementation is described by the pseudocode in algorithm 3. It uses the detected feature descriptors from each input image and appends them to a descriptor list vector. Using all of the descriptors from each input image, K-means clustering is used to create a dictionary of the image descriptors. A KMeans object is used with the default value of 10 for the number of times the k-means algorithm is run on different seeds. K-means clustering is computed for the feature descriptor list and the visual words vocabulary is created using the cluster centers. A default k value of 300 was chosen for each of the feature techniques implemented as quantizing the descriptors into 300 visual words provides enough information for the SVR without negatively affecting its computational performance. $k = 300$ was used in prior work which provided motivation for the quantization size [Raskar *et al.* 2015; Xiao *et al.* 2010]. Once the visual words vocabulary is created, the same feature detection method is used on an input image to extract its feature descriptors. In order to assign the descriptors to their specific vocabulary, vector quantization (vq) is used, which compares the descriptor with the centroids contained in the visual words dictionary and assigns it to the closest centroid. This process creates a frequency histogram specific to the input image which drastically reduces the amount of data contained in the feature vector.

3.3.5 Support Vector Regression

Before the SVR model can be created, the level TrueSkill scores and corresponding level feature vectors for each population level need to be aligned. The feature vector dataset and TrueSkill score dataset are merged together and then split into training data, $\boldsymbol{X}$ and $\boldsymbol{y}$, which is used to train the SVR model.

Algorithm 3 BOVW Feature Detection

Require: $imgs \leftarrow$ List of images
Require: $method \leftarrow$ Feature extraction method to use (*str*)
Require: $k \leftarrow$ Number of clusters
$descriptor\_list \leftarrow []$
$feature\_vectors \leftarrow []$
for i in $imgs$ **do**
    Load in image i
    Extract keypoints and descriptors of i using $method$
    $descriptor\_list.extend(descriptors)$
end for
$vis\_words \leftarrow kmeans(k, descriptor\_list)$
for j in $imgs$ **do**
    Load in image j
    Extract keypoints and descriptors of j using $method$
    $img\_hist \leftarrow create\_bovw(descriptors, vis\_words, k)$ ▷ Create bovw
    Add img_hist to $feature\_vectors$
end for
Scale $feature\_vectors$
Return $feature\_vectors$

For initial testing of the SVR model, the preprocessing function split the data into a testing and training dataset with a ratio of 0.3. This approach is commonly used as it provides a way in which to train a model on some data, the training set, and test it using unseen data, the test set. It also provides a way in which to calculate metrics, such as Root Mean Squared Error (RMSE), which estimates a model's accuracy. For this research, the design choice was made to not split the training data, as the more training data the better the model should perform on unseen data which will be more beneficial to the research being conducted.

Based on the research conducted by Hsu *et al.* [2003] it was found that correctly scaling the training dataset for SVM type models before being applied to the model greatly improves performance as it prevents larger numeric ranges from dominating the smaller ranges. To avoid this potential problem, a design choice was made to scale the feature data into a range between [0, 1] before storing the feature data. This range is one of the recommended ranges by Hsu *et al.* [2003].

To create the model, a base SVR object is created first, and then fitted to the pre-processed training data being passed into the function, $\boldsymbol{X}$ and $\boldsymbol{y}$. The initial fit of the data to the SVR in most cases is not optimal causing a poor SVR model to be created. In order to improve this model so that it can accurately predict TrueSkill scores for unseen levels based off of level feature vector data, the hyperparameters for the model need to be optimised.

In order to optimise parameters of an SVR a grid search is performed followed by a cross validation step, as stated in Section 2.3.2. The choice of the parameter grid was an important design decision because having a grid too large will negatively impact the time taken to optimise the model, whilst a grid too small may not contain optimal parameters.

The predicted output of the SVR model can directly be used in a fitness function, referred to as the human fitness function. This predicted output will be a TrueSkill score for a video game level and will allow it to be used as an additional objective for which the GA can optimise.

3.3.6 Fitness Functions

The fitness functions that are passed into the GA have been mentioned briefly in previous chapters. This section plans to expand on the fitness functions used and how they are calculated as they are an important part of the research being conducted. For each of the fitness functions, the objective of the GA is to maximize them.

Solvability

Solvability in a video game environment is a binary question with an answer of either yes, a level can be solved, or no, a level cannot be solved. The solvability fitness function in this implementation outputs a 0 for unsolvable levels and a 1 for solvable levels. As an example, assume there is a *Mario* level that has a pillar of unbreakable blocks with no possible way over or through them which are blocking the player from reaching the finish line. This would be considered an unsolvable level, thus the level will be given a solvability fitness score of 0. The fitness function determines the solvability of a level by using the A^* search based method by Hart *et al.* [1968].

Human Fitness

The human fitness function is the dominant aspect of this research as everything that has been developed is used in order to create a fitness function that provides a human element into the GA. Algorithm 4 shows the pseudocode for some of the $HumanFitness$ class which calculates a human fitness for an input level; the important function being $calc\_fitness()$. One other function, $set\_model()$, is not shown, but essentially updates the $model$ variable of the class to the model created from the population data. The $final\_answer$ variable is what is passed back into the GA which is the human fitness objective the GA is required to optimise. As mentioned in Section 3.3.3, lines 6 and 7 show how the clamping is performed if a negative TrueSkill score for the input level is predicted. Algorithm 5 is called within algorithm 4 and is how the SVR model actually predicts a TrueSkill score for an unseen level. To do this, it must first extract the feature vector for the input level and then use the trained SVR model to predict a TrueSkill

score. For the case when a model has not yet been created, lines 1 and 2 in algorithm 5, the function outputs a random score between 0 and 1 so that the initial population levels will be randomized. Without this step, each level in the initial population is the exact same which negatively affects each step in the PCG loop, Figure 3.1. It is worth nothing that optimising $trueskill\_calc()$ provides the highest increase in the GA's speed performance as it is a step which is called multiple times within the GA.

Algorithm 4 $calc\_fitness()$ ← function in HumanFitness class

Require: $levels$ ← List of List of Levels
1: $final\_answer \leftarrow []$
2: **for** $level\_group$ in $levels$ **do**
3: $total \leftarrow 0$
4: **for** $level$ `in` $level\_group$ **do**
5: $trueskill\_for\_level = trueskill\_level(level)$
6: **if** $trueskill\_for\_level < 0$ **then**
7: $trueskill\_for\_level = 0$ ▷ Clamping for negative predictions
8: **end if**
9: $total+ = trueskill\_for\_level$
10: **end for**
11: Append $total/len(level\_group)$ to $final\_answer$
12: **end for**
13: $final\_answer$ ← Return

Algorithm 5 $trueskill\_level()$

Require: $level$ ← An input Level
1: **if** $model$ is None **then**
2: Random uniform value between 0, 1 ← Return
3: **end if**
4: $feat\_vect \leftarrow generate\_features(level)$ ▷ Generate feature vector for $level$
5: $ts\_score \leftarrow model.predict(feat\_vect)$ ▷ Predict TrueSkill score from feature vector
6: Return ts_score ▷ Predicted TrueSkill score

This chapter discussed the various steps and design choices that are to be implemented in order for the goals of this research to be achieved. Chapter 4 explores the experiments conducted, along with the results gathered using the approach discussed to assist in proving the hypothesis of this research.

Chapter 4

Results and Analysis

This chapter breaks down the details for each of the experiments conducted using the proposed method outlined in Chapter 3 along with the results obtained for the experiments. The results are analyzed and discussed.

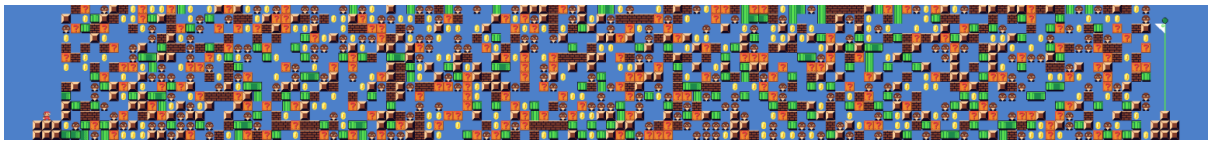


Figure 4.1: Example of a *Super Mario Bros* level of size (114×14)

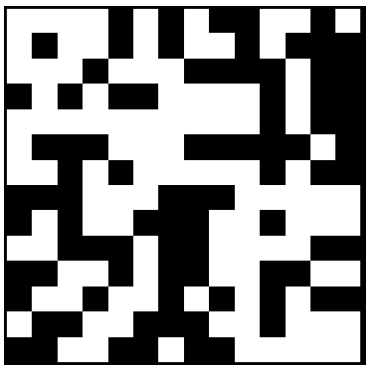


Figure 4.2: Example of a *Maze* level of size (14×14)

The experiments were conducted on two different 2D platformer video games, *Super Mario Bros*, shown in Figure 4.1, and *Maze*, shown in Figure 4.2. For each game, baselines were generated in order to show the effectiveness of each fitness function on the GA. The parameters used for each of the experiments for their respective games are discussed in their relevant sections.

4.1 Hyperparameter Optimisation

As discussed previously, there are parameters which can be optimised at various steps in the PCG loop. For the fitness functions, a weighted sum of fitnesses is used in order to capture multiple objectives as is done in prior work by Beukman *et al.* [2022a]. Thus, each individual weight determines how much influence the corresponding fitness function has on the GA. This means that fitnesses with low weights will still be considered by the GA, but at a reduced effectiveness compared to fitness functions with high weights. Human designers want levels that: (a) are solvable as unsolvable levels are not very useful, and (b) levels that contain human preferred features. This makes selecting the weights an important optimisation step for the fitness function. Equation 4.1 shows how the weighted sum is used in the overall fitness function, where α is the solvability weight and β is the human fitness weight.

$$\text{total fitness} = \alpha \times \text{solvability fitness} + \beta \times \text{human fitness} \tag{4.1}$$

Empirically, it was found that the value of the weights caused the GA to priorities one fitness function over the other. When $\beta > 0.39$, too much emphasis is placed on the human fitness causing the solvability fitness to be negatively affected. In order to ensure solvability was considered highly by the GA, an $\alpha = 10$ and a $\beta = 0.39$ was chosen.

In terms of the SVR and its performance, as discussed in Section 2.3.2, a grid search is used in order to optimise the parameters. The parameter grid used in the experiments conducted is shown in table 4.1. In order to verify that the selected parameters were optimal, a cross validation of size 5 is done on the parameter grid. Once the optimal parameter values are determined from the grid search, the SVR model is refitted to the training data with the newly optimised parameter values. This optimised model is what is passed back into the human fitness function of the GA and can vary between experiments due to the nature and size of the training data. An example of the performance of the SVR model before and after the optimisation step is shown in Section 4.2.3. The hyperparameter optimisation is a key step in the process, as the SVR exhibits poor predictive capabilities when default values are used.

Table 4.1: Grid search ranges for each parameter

Parameter	Grid Search Range
C	{0.001, 0.01, 0.1, 1, 10, 100}
γ	$\{auto, scale\}$
kernel	$\{linear, rbf\}$
ϵ	{0.1}

The default TrueSkill parameters values were used for μ and σ^2 so that the Gaussian distribution can be described as $N(25, \frac{25}{3}^2)$. Due to the pairwise comparison tournament style being a round robin, and the lowest population size being 20 in the experiments

discussed, enough games are played to ensure the correct skill is determined for each level, as discussed in Section 3.3.3.

4.2 Maze

Maze is a 2D platformer based game made up of two tile colours, black and white. The black tiles are considered 'walls' whilst the white tiles are considered 'empty'. The goal of the game is for the player to start on the top left white tile, coordinates (0, 0) and find a path along the white tiles to reach the end of the *Maze* located at the bottom right white tile, coordinates (14, 14). The player of the *Maze* may only move in the four cardinal directions. A *Maze* level is considered solvable if a connected path, of white tiles, exists between the starting tile and the end tile. A simple example of a 14×14 *Maze* is shown in Figure 4.3. A *Maze* can be scaled to any $N \times N$ size, however in each of the experiments discussed a *Maze* size of 14×14 is used as it provides a sufficiently large *Maze* for which the experiments could be conducted at reasonable speeds and test the proposed PCG loop thoroughly. A reason for choosing *Maze* is that it has been successfully used with genetic algorithms in prior work [Beukman *et al.* 2022ab].

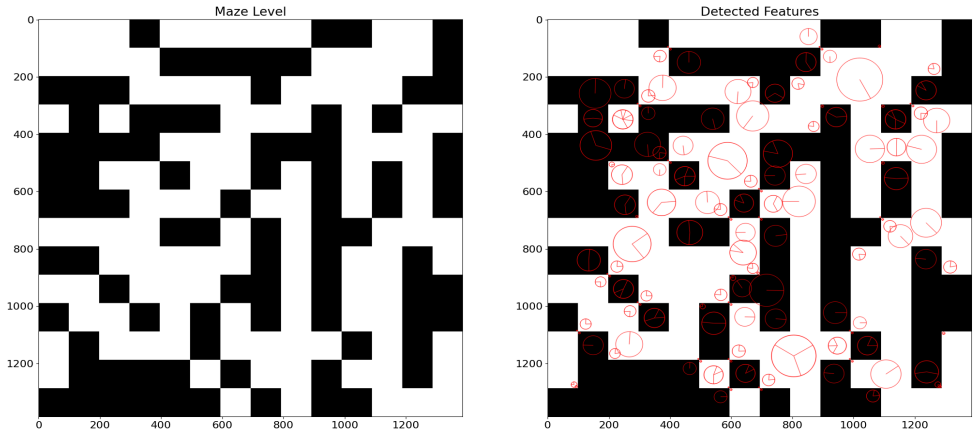


Figure 4.3: Figure showing a randomly generated *Maze* level, of size 14×14, followed by the same level overlaid with detected SIFT feature keypoints. Circle size indicates the scale of the keypoint and the lines indicate the keypoint orientation.

During initial experimentation for *Maze* each of the feature detection techniques were used; however, the SVR did not sufficiently predict unseen levels as was to be expected. It was determined that due to *Maze* only containing black and white tiles, the feature detection methods, in which a feature vector of size 300 was created, were unable to extract enough feature information which in turn caused the SVR to perform

Table 4.2: Table showing the binary version of the *Maze* level shown in Figure 4.3

1	1	1	0	1	1	1	1	1	0	0	1	1	0
1	1	1	1	0	0	0	0	0	1	1	1	1	0
0	0	0	1	1	1	1	0	1	1	1	1	0	0
1	0	1	0	0	0	1	1	1	0	1	0	1	1
0	0	0	0	1	1	1	0	0	0	1	1	1	0
0	0	1	1	0	1	1	0	1	0	1	0	1	0
0	0	0	1	1	1	0	1	1	0	1	0	1	0
1	1	1	1	0	0	1	0	1	0	0	1	1	1
0	0	1	1	1	1	1	0	1	0	1	1	0	1
1	1	0	1	1	1	0	0	1	0	1	1	0	0
0	1	1	0	1	0	1	0	1	0	1	1	0	0
1	0	1	1	1	0	1	1	1	1	0	1	0	1
1	0	0	0	0	1	0	1	1	0	1	1	0	1
0	0	0	0	0	0	1	0	1	0	0	1	1	1

poorly. As an example, Figure 4.3 shows the features detected, red circles, from a *Maze* level with SIFT is used. As can be seen in Figure 4.3, there are very few features detected, and the ones that are in arbitrary locations. Due to the small size of the *Maze* levels, LBP did not perform well as the amount of feature data was too small to train an accurate SVR model. In order to improve the SVR performance, each *Maze* level was converted into a binary grid, of size 14×14 which is shown in table 4.2, and flattened into a feature vector of size 1×196. The representation of the *Maze* level in table 4.2 is the binary version of the level shown in Figure 4.3. Using the actual level data, rather than feature extraction techniques, proved far more effective and is used throughout the *Maze* experiments.

As stated previously, virtual humans are used in order to specific which aesthetic features to prefer. For *Maze*, the virtual human is specified to prefer levels that contain black pixels on its border. One of the motivations for this preference is that in prior work, Beukman *et al.* [2022a], solvability in *Maze* levels was a result of a pathway existing around the border of the level, which caused the level to be solved without ever entering into the centre of the *Maze*. This does not result in a challenging level, thus having a preference that forces the pathway into the center of the *Maze* results in a better overall level.

4.2.1 Baseline Human Fitness

Each of the experiments run in this section test using only the human fitness function. This is achieved by setting $\alpha = 0$ such that solvability is not taken into account by the GA.

Figure 4.4 shows the results obtained from experiments in which a set number of GA iterations of 100 was used for a range different initial population sizes. For each population size, 5 different seeds were run and the averages of the seeds were used to calculate the average human fitness function value, shown in orange, and the average number of black border pixels, shown in green. The standard deviation for each value are shown as black lines. It is evident that as the population size increases, so does the average human fitness along with the number of black border pixels. For low population sizes, the number of black border pixels is at its lowest, this is due to small amount of training data for which the SVR is trained on. As the initial population size increases, so does the amount of training data for the SVR which in turn increases the number of black border pixels and human fitness due to the SVR performing better as it has been trained with more data. The number of black border pixels however does slow down to a stable value of around 40 at a population size between 200 and 500, whilst the human fitness continues to grow. This is due to the SVR performing well due to the large amount of training data, but due to the low number of generations the GA is not able to significantly increase the number of black border pixels. It was found, and discussed in Section 4.2.3, that increasing the number of GA iterations allowed for the human fitness function to be optimised which increased the number of black border pixels.

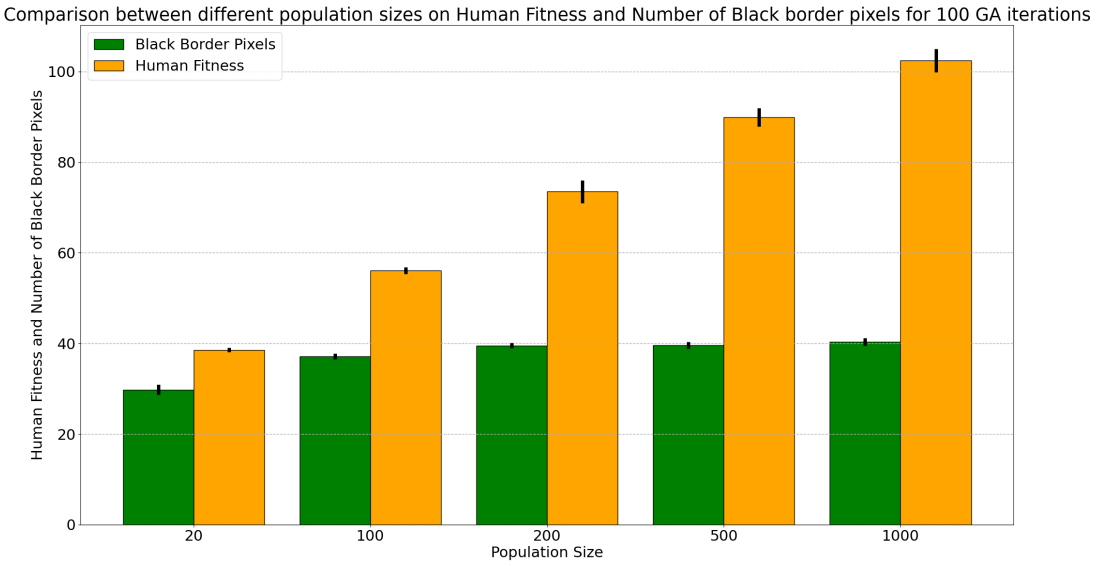


Figure 4.4: Graph showing the comparison between low and high initial population sizes and their effect on the average human fitness and average number of black border pixels on the best generated level for 100 iterations; black lines indicate standard deviation.

4.2.2 Baseline Solvability Fitness

In this experiment, only the solvability is used as the fitness to optimise by the GA. The experiment compared the average solvability to the number of GA iterations run and the effect optimising only solvability has on the average number of black border pixels.

Figure 4.5 shows the results obtained when a varying number of GA iterations is used. For each iteration size, 5 seeds were run and averaged to determine how a varying number of iterations affects the level solvability, shown in blue, along with the number of black border pixels, shown in green. Since human fitness is not being optimised, it is expected for the average number of black border pixels to not significantly change, which is the case from the results obtained. It is clear from Figure 4.5 that as the number of iterations increases, so does the average solvability whilst its standard deviation decreases. This shows that for higher iterations, the GA is able to generate more solvable levels than unsolvable levels. For 20 iterations, only around 36% of the levels generated were solvable. When looking at the number of black border pixels the value remains relatively low, with a maximum of 26 for levels in which the majority were not solvable. This indicates that in most cases, a level is solvable when the number of black border pixels is lower than 46% of the total border pixels.

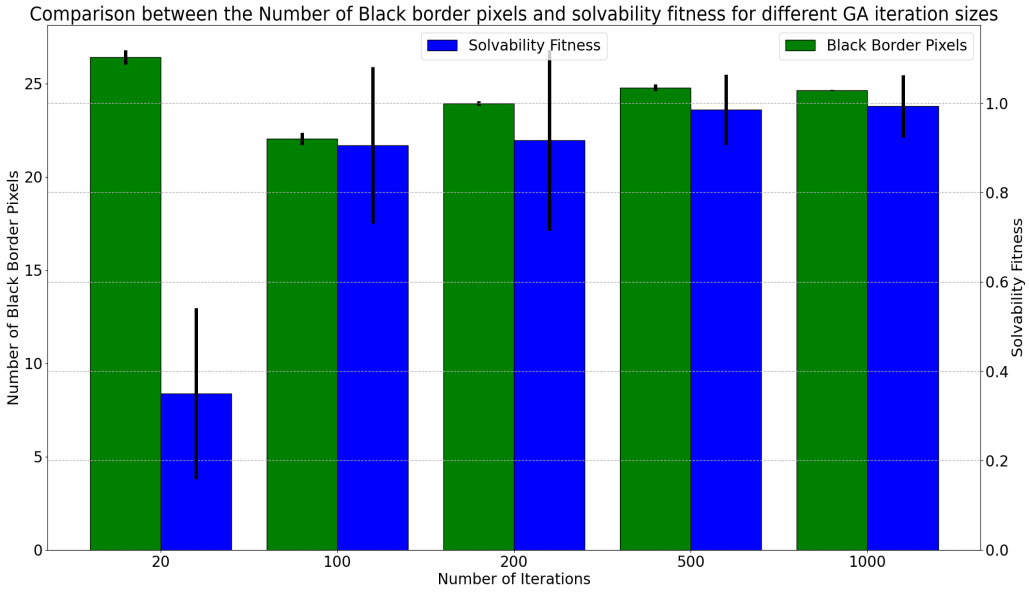


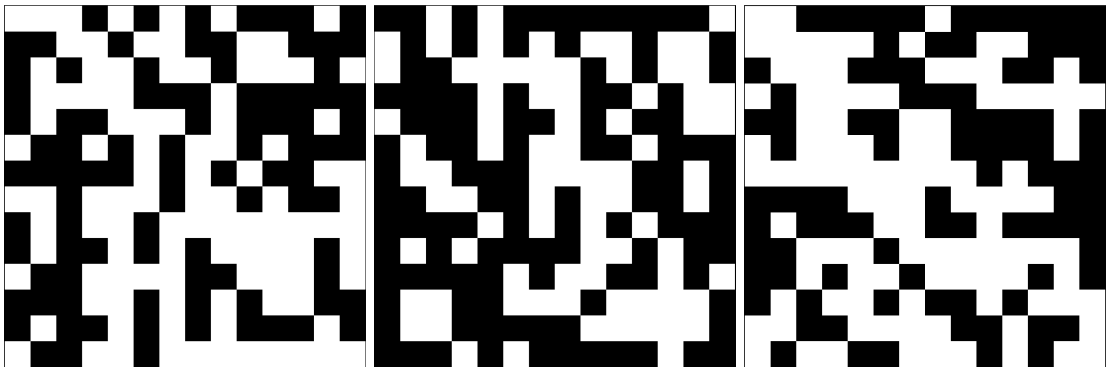
Figure 4.5: Graph showing the comparison between low and high GA iterations and their effect on the average solvability fitness and average number of black border pixels for the best generated level; black lines indicate standard deviation.

4.2.3 Maze Results

The majority of experiments conducted were on the combined fitness function for solvability and human fitness as this is the main focus of the research being conducted. For

each of the experiments, a solvability fitness weight of $\alpha = 10$ is used whilst a human fitness weight of $\beta = 0.39$ is used.

Figure 4.6 shows three different *Maze* levels, each of which have been generated in a slightly different manner. The first level in the figure, (a), was generated using only the solvability fitness where it can be seen that the leve is solvable, but only contains 25 black border tiles out of a total 56 tiles. The second level, (b), was generated using only human fitess which is why the level is not solvable. It contains far more black border pixels, being 40 black border pixels which is 71.43% of the total border pixels showing that the human fitness by itself does bias the generated levels towards more black border pixels. The last level, (c), is generated using both the solvability fitness and human fitness. The level is both solvable and contains more black border tiles, 32 out of 56, compared to the level shown in (a) which only focused on solvability. This brief example is an indicator that the combined fitness does infact allow for a solvable level whilst having a bias towards the preferred human features in a level. This section will go into the results obtained from experiments conducted when both the solvability fitness and human fitness are being optimised by the GA.



(a) (b) (c)

Figure 4.6: (a) *Maze* level generated using only the Solvability fitness function (25 black, 31 white). (b) *Maze* level generated using only the Human fitness function (40 black, 16 white). (c) *Maze* level generated using both Solvability and Human fitness functions (32 black, 24 white).

An important point to note about *Maze* levels and the virtual human preference of black border tiles, is that the start and end point for a solvable *Maze* level requires the first tile, coordinates (0,0), and the last tile, coordinates (14,14), to be white. Due to this, there is a contradiction as these tiles need to be white in order for a level to be solvable, but the virtual human preference would rather they be black tiles. Due to this contradiction and the need for solvable levels, a design choice was made to ensure that each the top left tiles, coordinates (0,0), (0,1) (1,0), and the bottom right tiles, coordinates (14,14), (13,14), (14,13), were always white tiles. By implementing this requirement, it always provides the *Maze* a start and end point, as well as a way in which to enter the centre of the *Maze* due to the *Maze* movement restrictions of only

the cardinal directions. This change does negatively affect the total number of black border pixels; however, it was considered a tradeoff worth making in order to improve the solvability of the levels being generated.

Maze Population Sizes

The experiments discussed in this section were conducted in order to determine what effect the initial GA population size has on the predictability of the trained SVR model for unseen levels. Since better prediction should result in a better human fitness function, this experiment also determines if higher initial population sizes increase the average number of black border pixels in subsequent levels generated by the GA.

Each of the subplots in Figure 4.7 provide detail into the effect the population size has on the predictability of the SVR model. The subplots (a, c, e, g, i) show the SVR model before the grid search has been used to optimise it, whilst subplots (b, d, f, h, j) show the SVR model once it has been optimised with grid search and a cross validation size of 5. The plots were generated using a test train split with a test split size ratio of 0.3. As stated previously in Section 3.3.5, the test train split is only used for better understanding of the model and no splitting of the data is done when running the PCG loop as a whole.

From each of the subplots in Figure 4.7 the population size ranges from 20 to 1000, and it is clear to see that a low number of population levels does not provide enough training data to the SVR for it predict the TrueSkill for unseen data accurately. For large population sizes, such as 500 and 1000 as shown in subplots (h) and (j) respectively, the SVR model performs very well showing that more training data does significantly improve the SVR model.

Each experiment is run for 100 GA iterations and a varying population size which ranges from 20 up until 1000. The results from the experiments are averaged over 5 seeds and the standard deviation between the seeds is shown as the shaded area in each graph. The value of 100 was chosen for the GA iterations as the solvability for the generated levels requires a minimum of 100 iterations in order to achieve a 90% solvability, as shown in Figure 4.5. In each graph, the baseline solvability and baseline number of black border pixels is shown by a dotted line and they are used to indicate an upper bound. This is to provide a comparison between when the GA only optimises a single fitness function and when the GA optimises multiple fitness functions, as the upper bound will be the maximum possible value when multiple fitness functions are optimised. In each of the figures, we are most concerned about the green line, number of black border pixels, orange line, human fitness, and dark blue line, solvability fitness. As to be expected, the average number of black border pixels is less than that of the baseline; however, the solvability is reached in each case making the slight decrease in average black border pixels worth the levels being solvable.

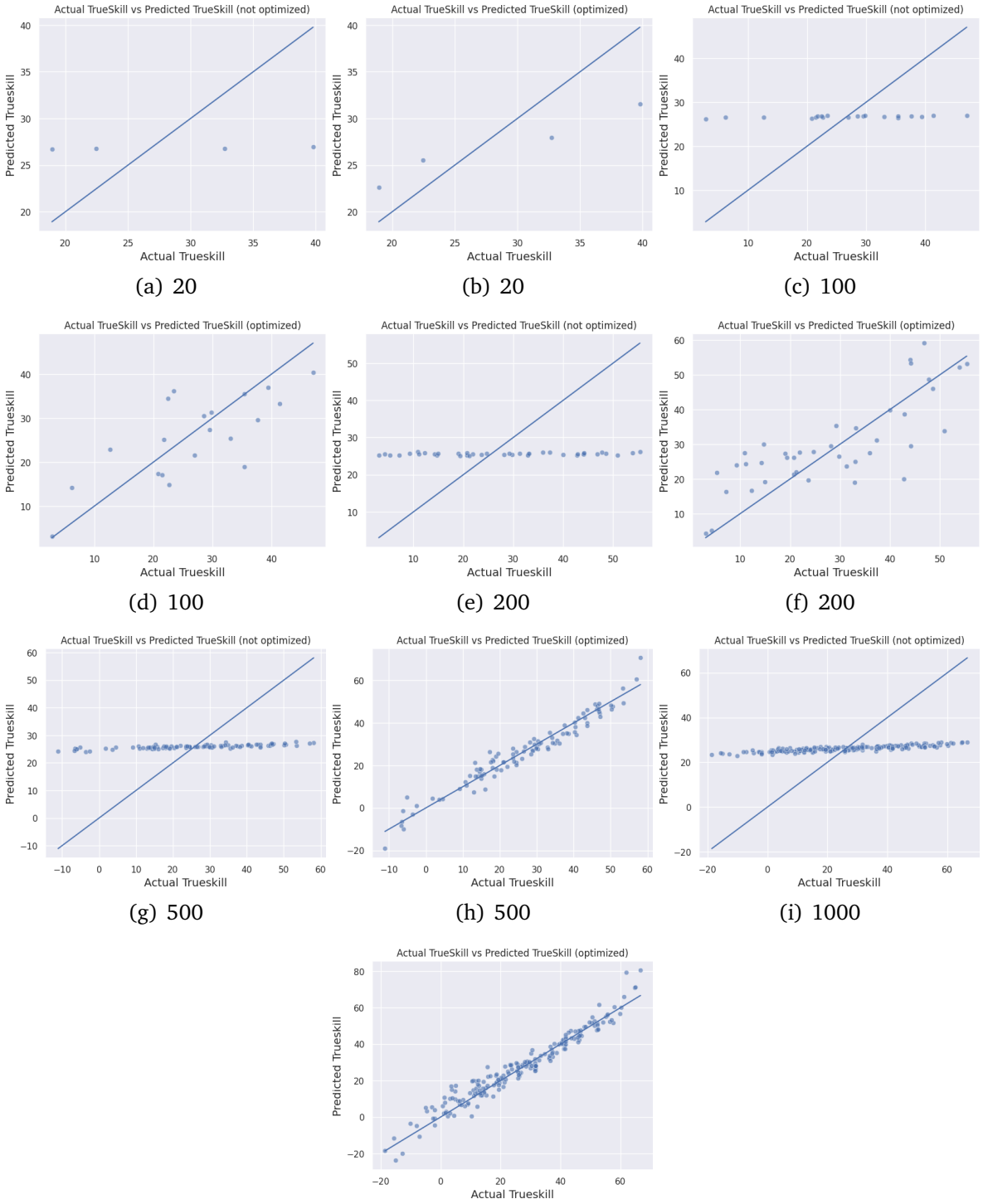


Figure 4.7: (a, c, e, g, i) SVR model prediction before optimisation; (b, d, f, h, j) SVR model prediction after optimisation.

Figure 4.8 shows how the GA performs with a population size of 20. It can be seen that although the solvability for each generated level is constantly at a value of 1, the human fitness and number of black border pixels fluctuates at low values. The average

number of black border pixels is around 23 which is only 41% of potential border pixels. This low number of black border pixels is caused by the SVR not receiving enough training data due to the low population size, an example of which is shown in subplot (b) of Figure 4.7, which causes the prediction of the TrueSkill scores to not be accurate.

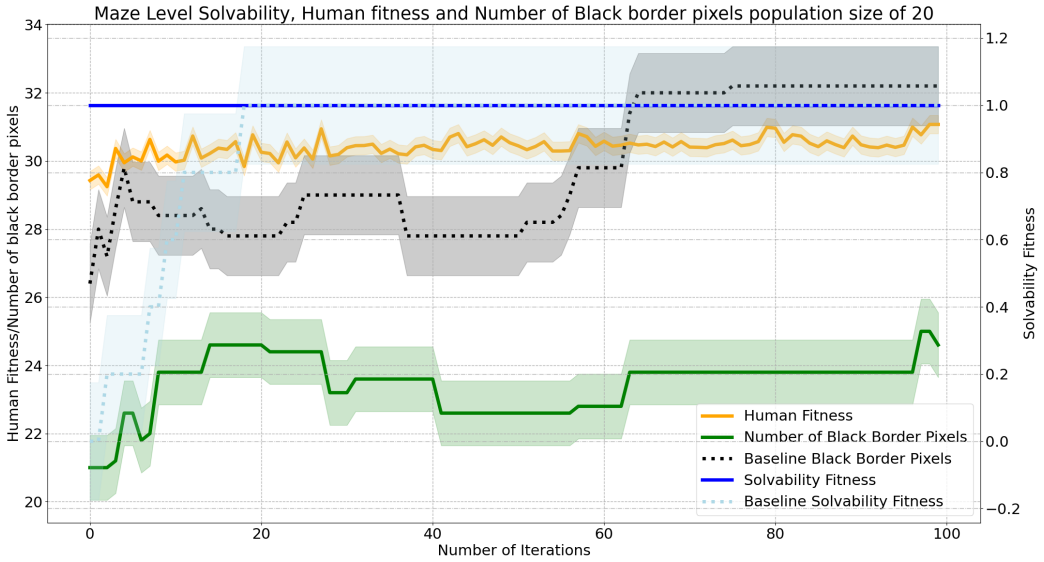


Figure 4.8: Figure showing solvability fitness, human fitness and number of black border pixels averaged over 5 different seeds for 100 GA iterations with an initial population size of 20; baseline plots are indicated by dotted lines; shading indicates standard deviation.

By increasing the population size to 100, as shown in Figure 4.9, there is an immediate increase in both human fitness as well as the average number of black border pixels. Along with this increase, the fluctuations have become less frequent showing that the GA is slowly biasing towards levels with black border pixels.

For a population size of 200, the solvability starts to fluctuate away from 1 for the first couple of iterations, as shown in Figure 4.10. The number of black border pixels and human fitness however are higher at a lower number of iterations, where the average number of black border pixels is around 31, when compared to previous results that used lower populations. These values continue to increase as the number of iterations increase.

Figure 4.11 shows that larger initial population sizes result in an increase to average black border pixels at lower iterations, with an average of 33. At this population size however the solvability is negatively affected as a stable average of 1 is only achieved at around 38 iterations.

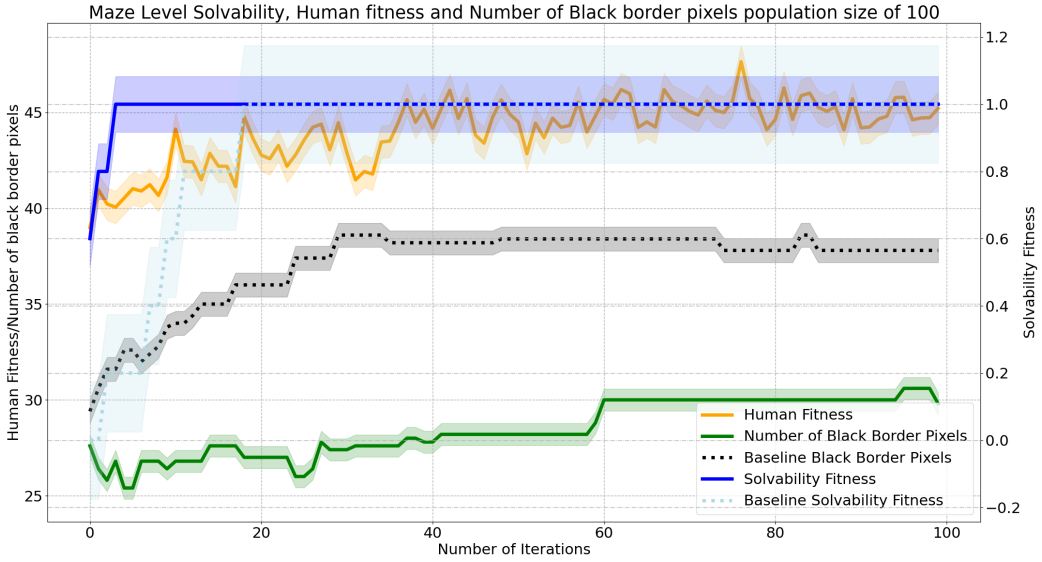


Figure 4.9: Figure showing solvability fitness, human fitness and number of black border pixels averaged over 5 different seeds for 100 GA iterations with an initial population size of 100; baseline plots are indicated by dotted lines; shading indicates standard deviation.

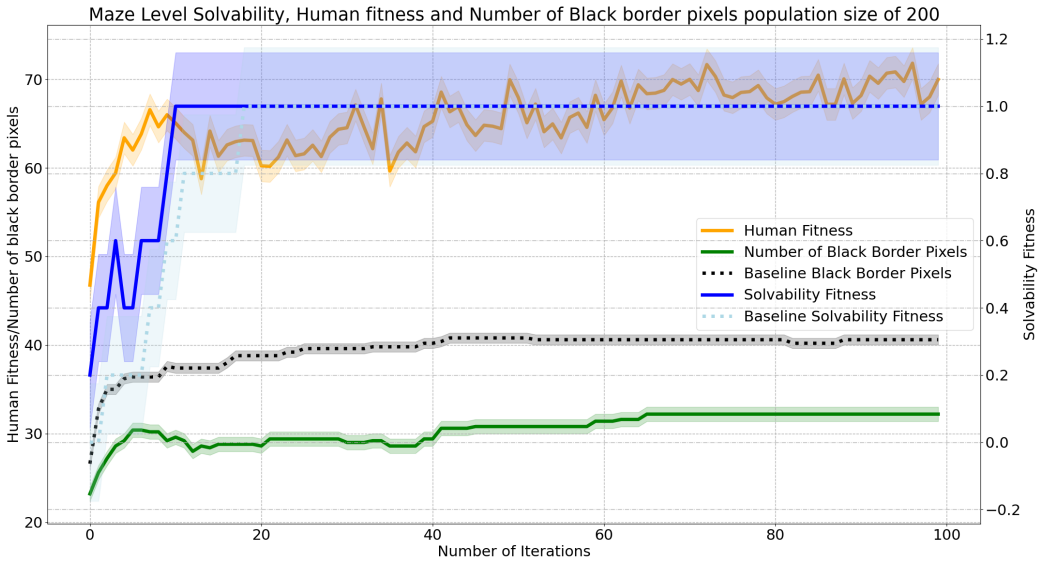


Figure 4.10: Figure showing solvability fitness, human fitness and number of black border pixels averaged over 5 different seeds for 100 GA iterations with an initial population size of 200; baseline plots are indicated by dotted lines; shading indicates standard deviation.

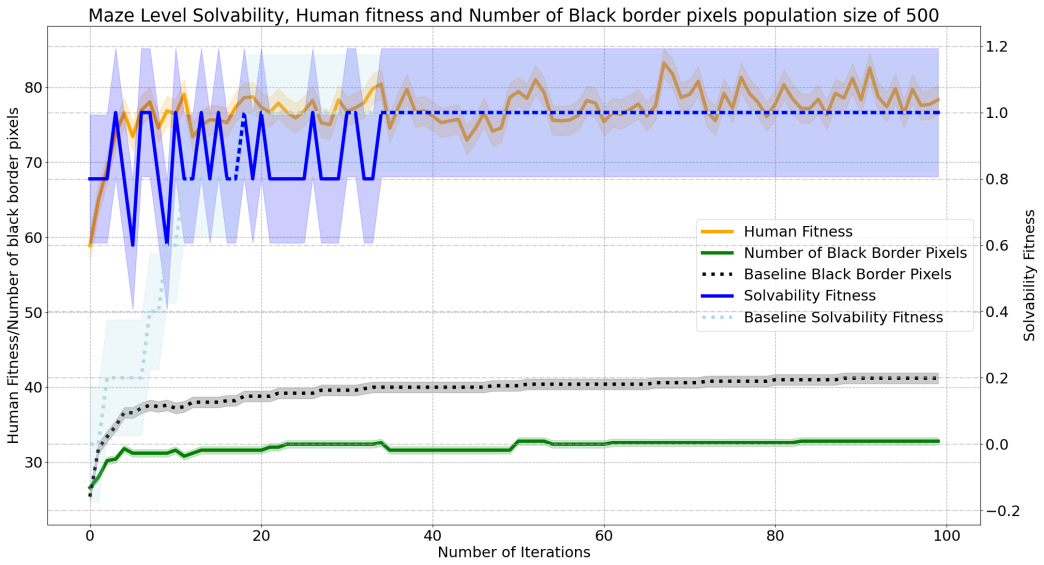


Figure 4.11: Figure showing solvability fitness, human fitness and number of black border pixels averaged over 5 different seeds for 100 GA iterations with an initial population size of 500; baseline plots are indicated by dotted lines; shading indicates standard deviation.

The last experiment for population sizes is shown in Figure 4.12. This is the worst performing result in terms of solvability, as the solvability fitness is unstable. This indicates that there is in fact diminishing returns for very large population sizes. The number of black border pixels has the highest average out of all the experiments at around 34 which is slightly higher than the average achieved with a population size of 500. The standard deviation in the number of black border pixels is the lowest out of all of the experiments, indicating that population sizes larger than 1000 for 100 GA iterations will not significantly increase the number of black border pixels.

In conclusion to the the results from the experiments which are shown in figures 4.8 to 4.12 and the SVR model results shown in Figure 4.7 it can be determined that initial population sizes between 200 and 500 perform the best for generating levels that have a bias towards levels that contain the virtual human preference as well as maintaining the solvability of the levels. For population sizes lower than 200 the SVR model does not perform well enough for accurate predictions which results in the human fitness being optimised towards the wrong features. Although the SVR model performs very well for population sizes larger than 500, there is too much of a bias towards the virtual human preference that it negatively affects the solvability of the level. Very large population sizes thus cause diminishing returns.

Figure 4.12: Figure showing solvability fitness, human fitness and number of black border pixels averaged over 5 different seeds for 100 GA iterations with an initial population size of 1000; baseline plots are indicated by dotted lines; shading indicates standard deviation.

Maze Iteration Comparisons

In the previous experiment, we investigated the effect of varying the initial population size. Now we look how the number of GA iterations affects the number of black border pixels for an initial population size of 200. The initial population size of 200 was chosen due to it performing equivalently in terms of average number of black border pixels to a population size of 500 and 1000 when only the human fitness function was investigated as shown in Figure 4.4. A population of size 200 also provides enough training data for the SVR model to perform well, which is shown in Figure 4.10 and subplot (f) of Figure 4.7.

Looking at the results shown in Figure 4.13, for a set population size, an increase in GA iterations increases the average number of black border pixels for generated levels. As the GA runs for longer iterations, it is given more opportunities to optimise the human fitness function, which it successfully does and it shows in the results obtained. The maximum average amount of black border pixels obtained was 35 at around 400 iterations. During initial iterations the average number of black border pixels increases quite rapidly and reaches a stable point close to 400 iterations. As such, iterations greater than 400 do not provide any benefit to the GA.

The results obtained however do not mean much if the levels being generated with higher black border pixels are not solvable. The blue line in Figure 4.13 shows the

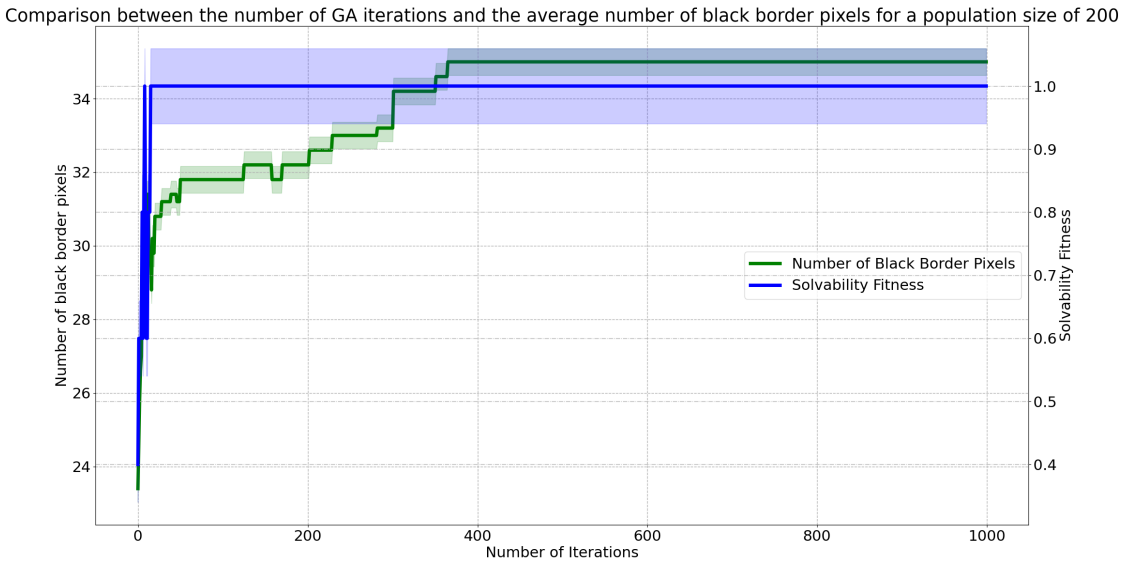


Figure 4.13: Figure showing how varying the number of iterations affects the solvability fitness and average number of black border pixels averaged over 5 different seeds for a population size of 200; shading indicates standard deviation.

corresponding average solvability. From the plot it is clear that solvability average is close to 1, after only a few iterations, which indicates that the majority of levels generated are in fact solvable. From these results it can be determined that level solvability remains achievable whilst the number of black border pixels increases as the number of GA iterations increase. The average number of black border pixels increases by 20% when human fitness is used in the GA for a population size of 200 after 400 iterations. The results obtained show that the GA is correctly generating features with a bias towards the preference of the virtual human. This shows that the method proposed can successfully integrate human advice as a fitness function to a multi-objective GA for video game levels whilst still adhering to standard level characteristics.

Multiple Human Inputs

The additional question proposed in Section 3.1, asks if using human inputs multiple times during the PCG loop increases the virtual human feature preference in the generated levels. This experiment aims to provide an answer. For the PCG loop to accommodate for multiple inputs, the approach discussed in Chapter 3, essentially occurs multiple times throughout the PCG life cycle which is shown in Figure 4.14.

In summary, for each time an input by the virtual human is required, the virtual human will perform pairwise comparisons on a new population that has been generated using the GA with the current iteration of the SVR model. These comparisons will result in new TrueSkill scores which will be fed into a completely new SVR model, along with

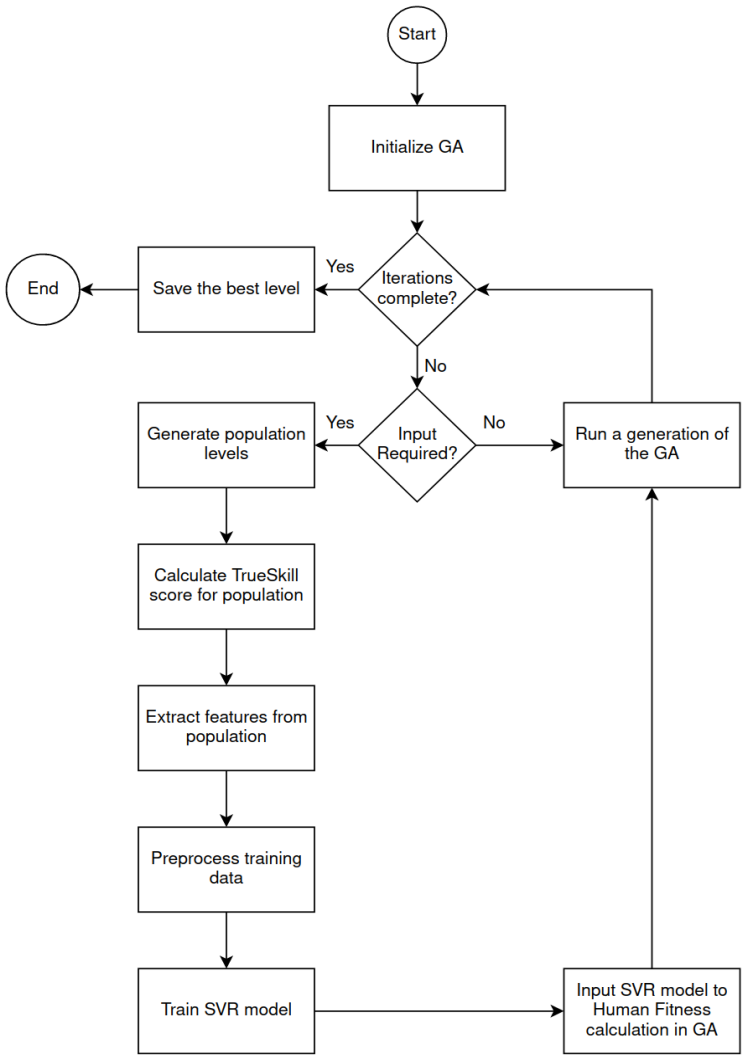


Figure 4.14: Flow diagram showing a high level overview of the experimental structure for the multi human input PCG process.

the corresponding level feature vectors in order to train the SVR model. This model is then passed back into the human fitness function so that the GA can continue to run for more iterations until the maximum number of iterations is reached or another input from the virtual human is required. Due to each new input running the entire research approach, the time taken to perform multiple ask increases significantly compared to a single ask implementation.

The experiment conducted was run on a population size of 200, for 200 GA iterations and asked for human advice a total of 3 times. Each input occurred at 0 iterations, 80 iterations and 160 iterations. As can be seen in Figure 4.15, the vertical dotted purple lines indicate at which point human advice input is provided. The experiment was run for a single seed due to the large amount of time required for the experiment to complete. The results show that the number of black border pixels increases from 28 to 34 after the second human input before settling at 32. For the third human input

the value decreases slightly, but improves back up to 32 again after the GA performs a few more iterations. It is important to note that the solvability remains at 1 throughout each human input. An interesting detail is that human fitness reduces by close to half, to a point where it is almost identical to the number of black border pixels. This is due to the introduction of a new model where the majority of the training data contains levels with a high number of black border pixels when compared to the original population. TrueSkill is a relative ranking system, so when the majority of the population is good, there won't be overly large values, such as the case after the second human input. As previous experiments found, an increase in iterations increases the number of black border pixels which is also evident from the results shown. Thus it can be determined that multiple human inputs does improve the bias towards the human preferred features, however further experiments will need to be conducted in order to optimise the performance.

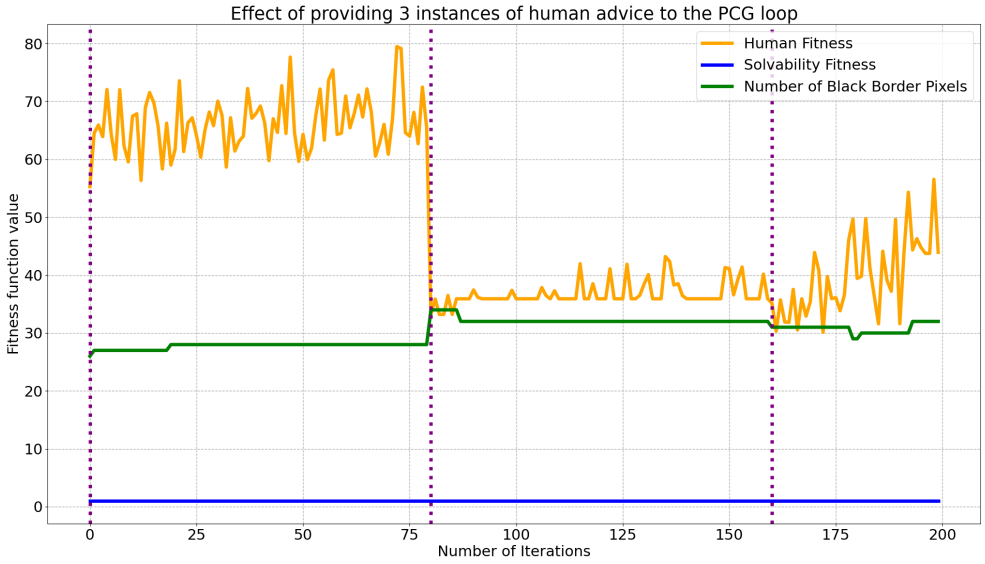


Figure 4.15: Figure showing 3 separate instances at which the virtual human inputs advice, shown as vertical dotted purple lines at x=0, x=80 and x=160.

Due to this research not focusing directly on adding multiple iterations of human advice into the PCG loop, the number of iterations, number of inputs and initial population size has not been optimised, but this could be a valuable question to answer in future work.

4.3 Mario

The second platformer game for which this research conducted experiments on is *Super Mario Bros*. SMB is a very popular 2D platformer video game where the object is to get

Mario from the start of the map, the left side, to the finish line flag, the right side of the map. Throughout the level however are enemies that must be avoided, and coins which can be collected. The version of the SMB implemented is a simplified version of SMB where the only enemies are mushrooms and no power ups are included. If power ups and additional enemies were to be included, the levels would become far more visually complex. This simplification limits the number of possible block types to nine. The collecting of coins is not taken into account in this research. *Super Mario Bros* was chosen as the second game as it has been used extensively in prior PCG based work [Beukman *et al.* 2022a; Ferreira *et al.* 2014; Shu *et al.* 2021a; Pedersen *et al.* 2009]

Each of the levels generated by the GA for SMB were of size 114×14. This size can be easily adjusted, but for these experiments it was kept constant in order for accurate comparisons to be made.

Figure 4.16 shows a randomly generated SMB level and corresponding features, red circles, which were extracted using SIFT. The number of features found by SIFT for a SMB level, compared to that of a *Maze* level shown in Figure 4.3, shows that SIFT performs far better on images that contain multiple different textures, rather than simple black and white blocks. A zoomed in look at the detected features is shown in Figure 4.17. There are SIFT features detected for almost every block in the level, showing that feature detection methods are a viable option for detecting features in 2D video game levels with multiple different textures.

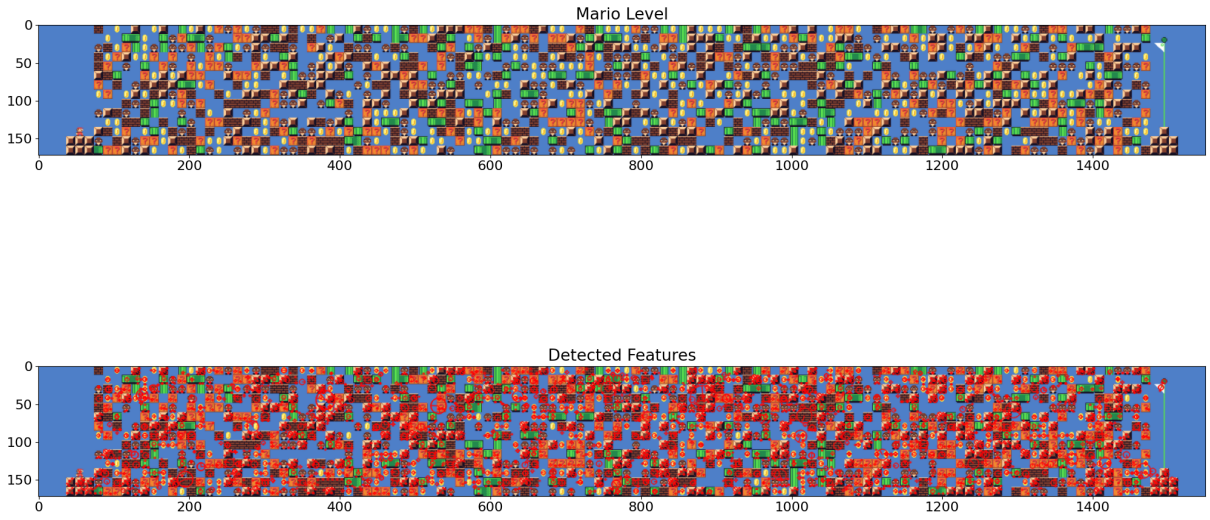


Figure 4.16: Figure showing a randomly generated *Super Mario Bros* level and the features detected on the level using SIFT.

For the experiments conducted on SMB, an entropy fitness function was included as an additional objective for the GA to optimise. Entropy is traditionally used to describe the measure of unpredictability of information content [Shannon 1948], but the work by Ferreira *et al.* [2014] found a way in which to use entropy to measure ground unpre-

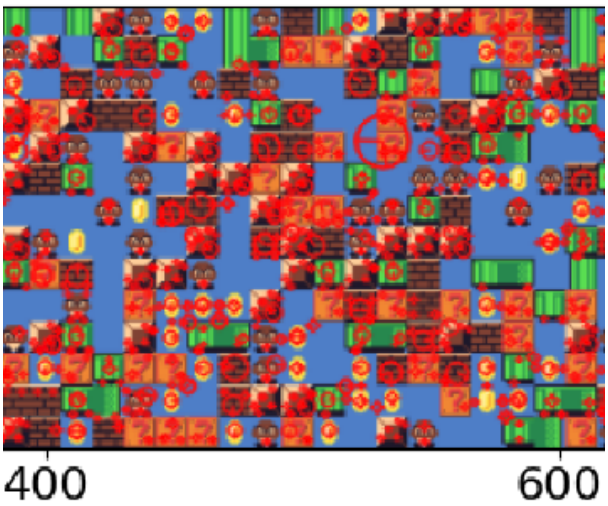
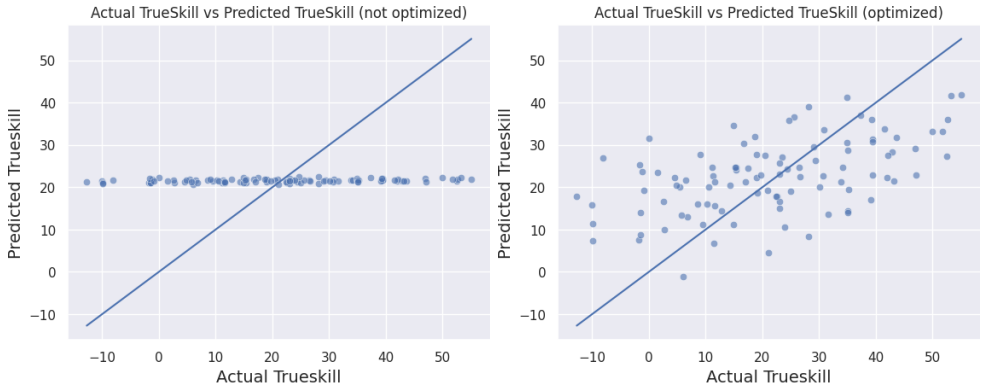


Figure 4.17: Figure showing a zoomed in section of the SIFT features detected in Figure 4.16. Circle size indicates how prominent the keypoint is and the lines indicate the keypoint orientation.

dictability for platform games. This design choice was made as the solvability fitness by itself did not provide much change to each of the generated levels. The entropy fitness used provides a way in which each block type can be evolved independently and is based off of the work by Ferreira *et al.* [2014]. In order to calculate the entropy fitness, a level is split into non-overlapping chunks where the average entropy for each chunk is returned. The value of the entropy fitness is just the value of the entropy itself which tries to stabilize at a value of 1. By including entropy as a fitness function, equation 4.1 is updated to form equation 4.2 where δ represents the entropy fitness weight.

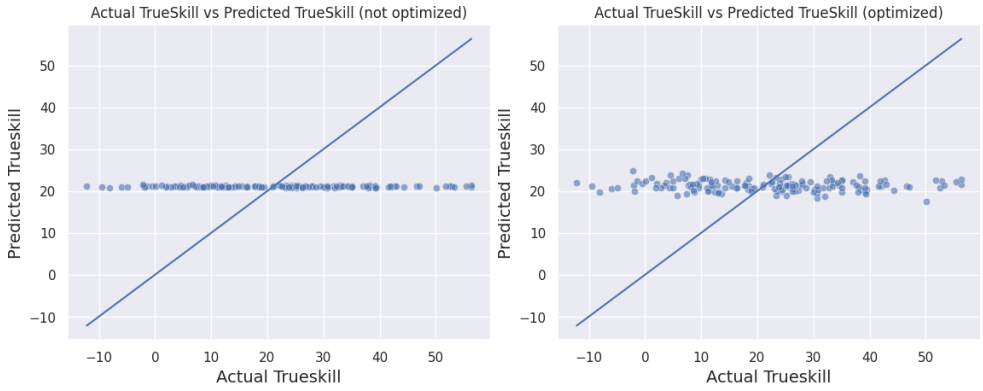
$$\text{total fitness} = \alpha \times \text{solvability fitness} + \beta \times \text{human fitness} + \delta \times \text{entropy fitness} \quad (4.2)$$

During the experiments for *Mario*, it was found that the approach of flattening the binary version of the level into a 1×1596 feature vector and using it as the input to the SVR did not perform as well as it did in *Maze*. The results shown in subplots (c) and (d) of Figure 4.18 show the non-optimised SVR, (c), and the SVR after optimisation, (d), that was generated using a population size of 500 and the flattened feature vector. It is clear that the results are very poor and the SVR did not perform at an accuracy that is required for human fitness to correctly be used in a GA. As such, the approach of using feature detection techniques was readdressed and it was found that they performed far better than the flattened vector. Subplots (a), no optimisation, and (b), optimised, of Figure 4.18 show the performance of the SVR when the training data is an input of features that are extracted using SIFT and quantized into a feature vector of size 300. Based on these results, SIFT is used to extract features from the *Mario* levels throughout the experiments. Although the SVR is not perfect, it performs well when enough training data is provided, making it a viable option to successfully be used for the human fitness functions.



(a) SIFT

(b) SIFT



(c) Flattened array

(d) Flattened array

Figure 4.18: (a) SVR model prediction before optimisation for a population size of 500 trained using SIFT feature vectors (b) SVR model prediction after optimisation for a population size of 500 trained using SIFT feature vectors (c) SVR model prediction before optimisation for a population size of 500 trained using a flattened feature vector of the level (d) SVR model prediction after optimisation for a population size of 500 trained using a flattened feature vector of the level.

Similarly to *Maze*, a virtual human is used in order to specify a preference of features in a *Mario* level. The feature preference used by the virtual human for *Mario* is the number of pipe blocks in the bottom row of the level, essentially the virtual human wants the ground blocks to be pipes. The motivation for this choice is that when solvability is run as a standalone fitness, each of the ground blocks are unbreakable blocks which makes the level very linear and easily solvable. Pipes blocks on the other hand are perfect for Mario as a character, since he is a plumber, which gives motivation to create a level where the majority of ground blocks to run over are pipes.

4.3.1 Baseline Human Fitness

In order to determine if the method proposed in Chapter 3 can be applied to SMB, experiments were conducted for a baseline to be created. Each of the baseline exper-

iments used only a human fitness function, where $\alpha = 0$ and $\delta = 0$, thus entropy and solvability will not affect the GA.

The results shown in Figure 4.19 show that the GA is correctly optimising the human fitness function as the population size increases, which is directly impacting the number of pipe blocks found in the bottom row of a generated *Mario* level. The result it to be expected as we know from the results obtained in *Maze*, that more training data improves the model which allows for the preferred human features to be better predicted. Although the number of pipe blocks does increase, the increase is rather small, with only a 10 pipe block increase compared to when human fitness is not optimised by the GA, as shown in the results discussed in Section 4.3.2. This low increase in pipe blocks can be related to the accuracy of the SVR model as well as the parameters not being fully optimised for *Mario*.

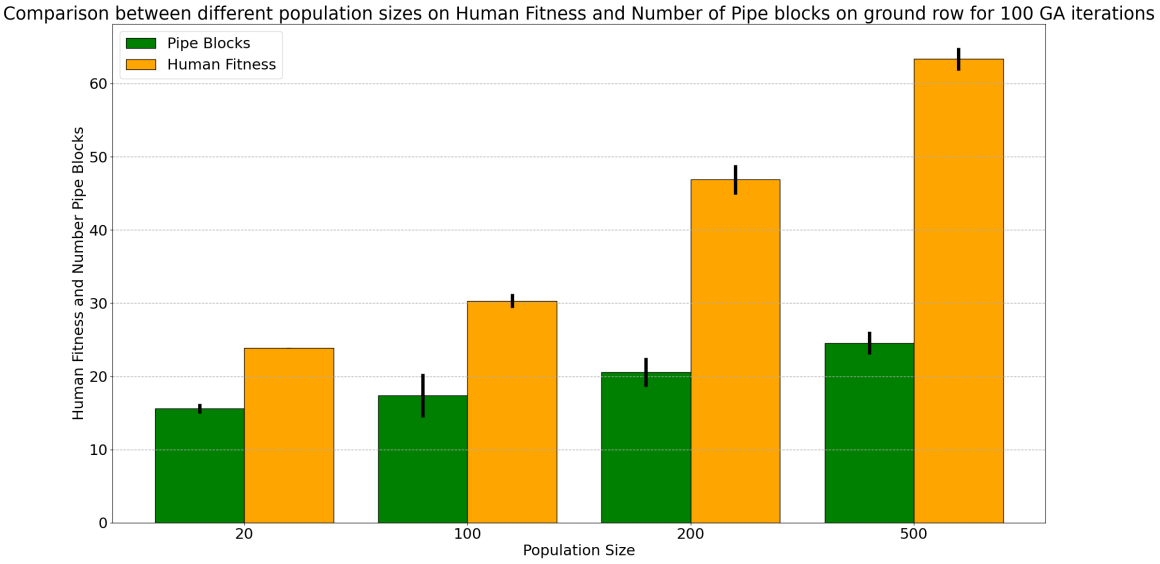


Figure 4.19: Graph showing the comparison between varying initial population size when only the human fitness is being optimised by the GA for 100 iterations; standard deviation is indicated by the black lines.

4.3.2 Baseline Solvability and Entropy

The purpose of this experiment is to provide a baseline model in order to see how entropy and solvability are affected by the number of GA iterations run. As such, the experiment used an entropy weight of $\delta = 1$ and a solvability weight of $\alpha = 10$. Additionally, this experiment provides a baseline to the average number of pipe blocks in the ground row when no human fitness is being optimised.

The results for the experiment are shown in Figure 4.20, where each experiment was run on 5 different seeds and the average standard deviation is show by the black lines. It can be seen that solvability remains at 1 for each set of iterations, with a very low standard deviation. This indicates that the *Mario* levels are in general far more solvable that *Maze* levels. The entropy increases significantly as the number of iterations go up, which shows that a minimum of 100 iterations is required in order for entropy to be maximised. Iterations larger than 100 does increase the entropy, but by a small margin. The average number of pipe blocks in the bottom row of the *Mario* levels are similar throughout the change in iteration size, where they range from an average of 12 to 16, which is to be expected as the GA is not actively trying to increase them.

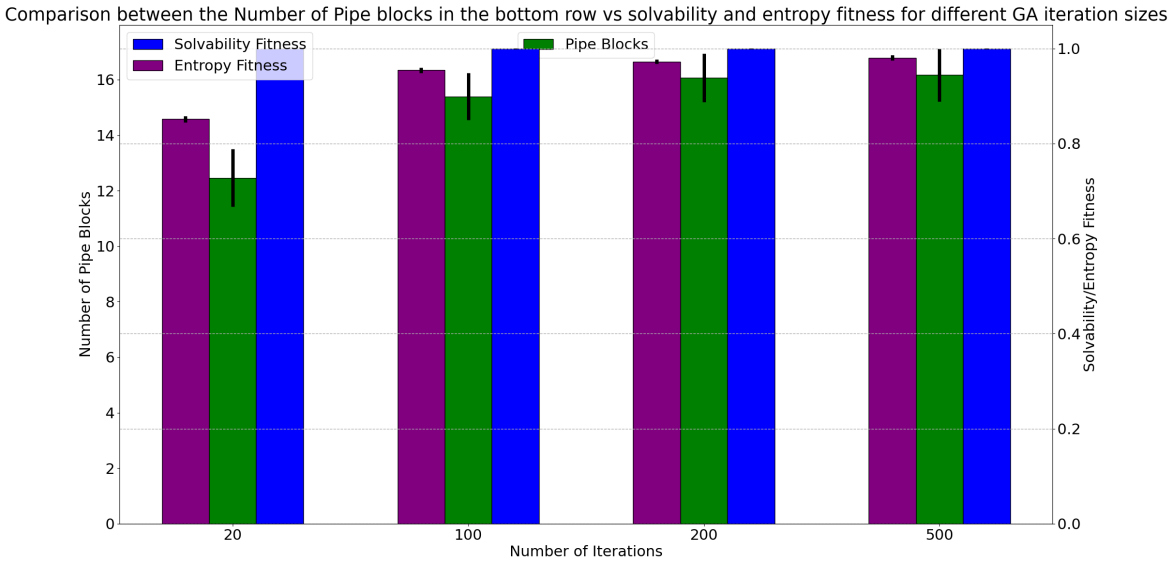


Figure 4.20: Graph showing the comparison between varying GA iterations and their affect on average solvability, entropy and the average number of pipe blocks in the bottom row of the best generated level of the iteration; standard deviation is indicated by the black lines.

4.3.3 Mario Results

For each of the following experiments, entropy, solvability and human fitness were optimised by the fitness function in the GA. Equation 4.2 provides the relationship of the weights and fitness functions, where the following weights are used in each of the experiments: $\alpha = 10$, $\beta = 0.39$ and $\delta = 1$.

Mario Population Sizes

The experiments conducted use a set number of iterations of 100 for a varying population size. Although a similar experiment was done for *Maze*, in Section 4.2.3, these

experiments are conducted to demonstrate that multiple different characteristics can be optimised, whilst the number of preferred features increases. Each of the experiments were run on 5 different seeds and the average was used, where the shaded areas indicate standard deviation. For each of the results for the experiments conducted the baselines are shown in dotted lines for solvability, light blue, entropy, yellow, and pipe blocks, black, and they each represent an upper bound. The upper bound is to show the value reached when optimising a single aspect of the GA, thus when optimising multiple values, they will unlikely reach the same value. The lines we care about are the human fitness, orange, number of pipe blocks in the ground row, green, solvability fitness, dark blue, and entropy fitness, purple. As stated previously, SIFT is used to extract the feature vectors from each *Mario* level, thus the experiment additionally demonstrates that standard feature detection techniques are a viable option for the method proposed when video game levels contain multiple different textures.

Figure 4.21 shows that using a small initial population of 20 levels did not provide decent results in terms of the number of pipe blocks as only a maximum of 13 were generated in the ground row of the level. This was to be expected, as discussed in Section 4.3.1, as the low population size causes the SVR model to incorrectly predict what the human preference was. As such, the human fitness function is at a constant 28, and is hidden under the solvability fitness line in the figure. The solvability and entropy perform well with solvability remaining at 1 whereas entropy performs slightly worse than the baseline.

As the population increases, the human fitness is being optimised better by the GA which is shown in Figure 4.22. The human fitness increases as the number of iterations increase and the number of pipe blocks are similar to those of the baseline. The levels remain solvable with no change in the solvability fitness whilst the entropy fitness fluctuations are reduced.

For a population size of 200, the human fitness is being optimised correctly to generate levels more inline with what the human preference is. As the iterations increase, so does the number of pipe blocks, showing that the SVR model is predicting higher TrueSkill scores for levels that contain pipe blocks on the ground row. As seen in Figure 4.23, the number of pipe blocks is very similar to the baseline, the only difference being the baseline generated levels with higher human preference features in less GA iterations. This shows that for a large enough population size, the proposed method can successfully optimise solvability, entropy and human fitness for a GA when using *Super Mario Bros* levels.

Figure 4.24 shows that a population size of 500 performs similarly to a population size of 200 in terms of the number of pipe blocks. The average solvability fitness begins to fluctuate between 0.8 and 1 showing that at large population sizes, the solvability of the levels generated is negatively affected. Similarly to what was found in the *Maze* experiments, large initial population values cause diminishing returns. Based on the

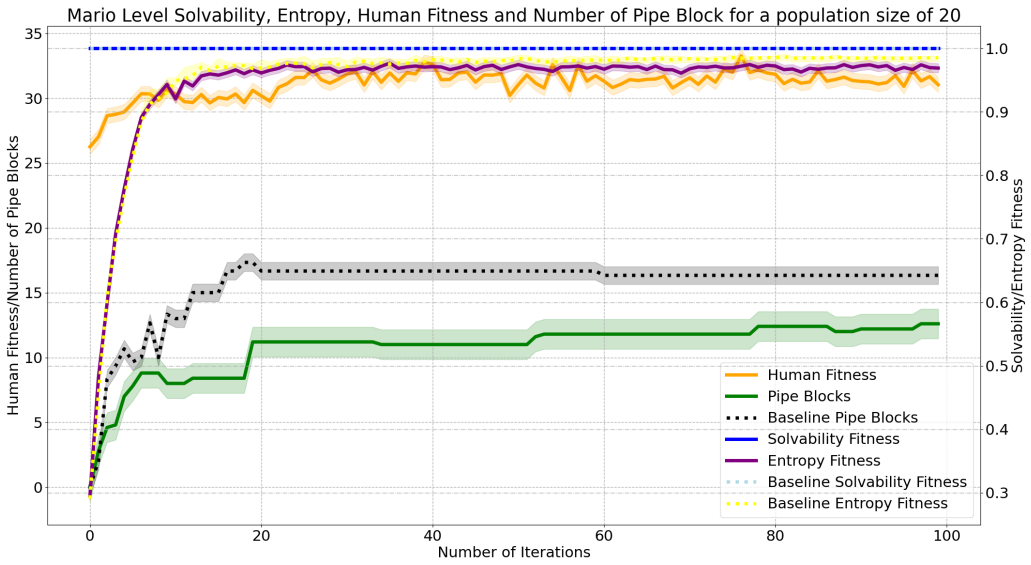


Figure 4.21: Figure showing solvability, entropy, human fitness and number of pipe blocks in the ground row of a *Mario* level for 100 GA iterations with an initial population size of 20, averaged over 5 different seeds; baselines for number of pipe blocks, solvability and entropy fitness are indicated by dotted lines; shading indicates standard deviation.

results shown, it is possible that if more features were to be extracted from each of the levels, the SVR could potentially benefit more from a smaller population size. For large population sizes, the model performance does not improve enough to tradeoff the non-stability of the solvability fitness.

Based on the results obtained from the experiments conducted, it can be determined that the proposed method for using human input as an additional objective to a multiobjective fitness function in a GA successfully works for *Super Mario Bros*. The proposed method can successfully improve the aesthetics of a level by generating features more inline with the preference of a human, whilst still adhering to other level characteristics such as solvability and entropy. The findings can still be extended, by potentially combining multiple feature detection techniques in order to extract a more detailed feature vector for the generated level which should improve the accuracy of the SVR model.

This chapter displayed and discussed the results obtained from each of the experiments run for both *Maze* and *Super Mario Bros*. It was found that the implementation to assign a numeric value to the aesthetics of a video game level is successful and can be input back into the fitness function of a GA to produce a new population of levels that have an increased average number of human preferred features, applicable to both *Maze* and *Super Mario Bros*. The initial population size and number of GA itera-

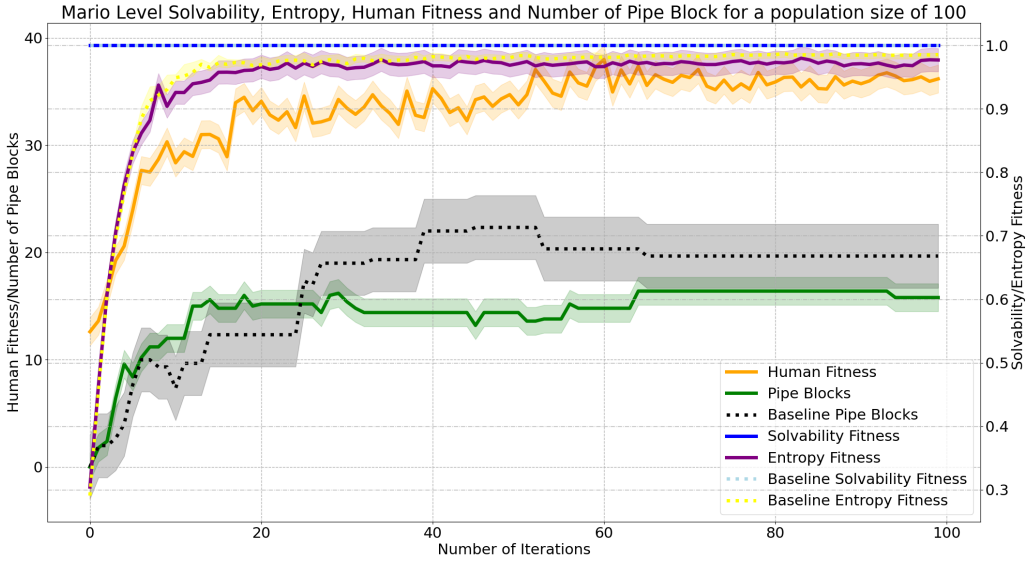


Figure 4.22: Figure showing solvability, entropy, human fitness and number of pipe blocks in the ground row of a *Mario* level for 100 GA iterations with an initial population size of 100, averaged over 5 different seeds; baselines for number of pipe blocks, solvability and entropy fitness are indicated by dotted lines; shading indicates standard deviation.

tions run directly affect the outcome of the proposed method, where higher population sizes are beneficial to the performance of the SVR, and higher GA iterations allow for the human fitness to be better optimised in future populations. Experiments were run to determine whether using multiple human inputs at various stages of the PCG loop would improve the PCG, although the results for multiple inputs is promising, further experiments need to be run in order for a concrete conclusion to be reached.

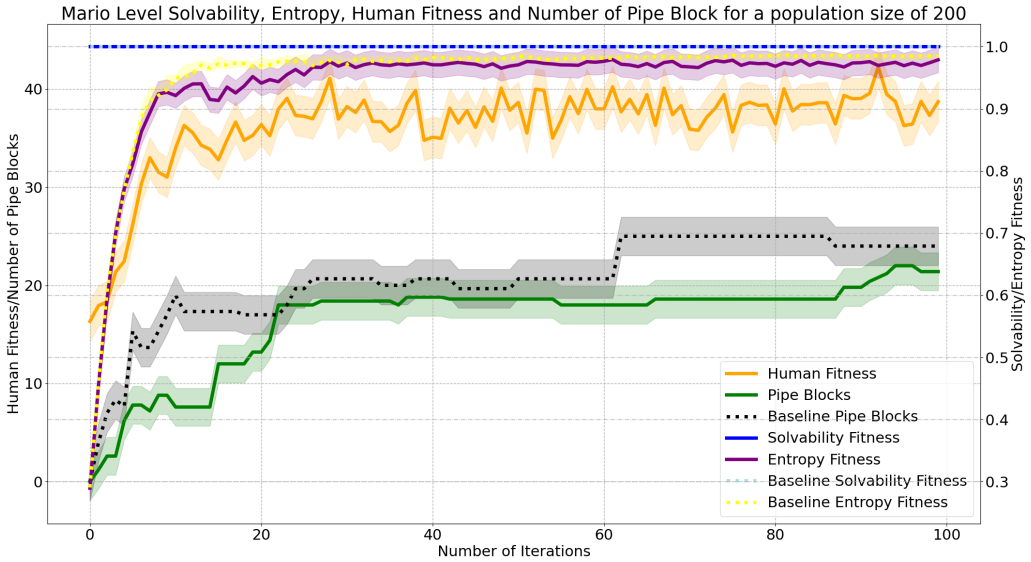


Figure 4.23: Figure showing solvability, entropy, human fitness and number of pipe blocks in the ground row of a *Mario* level for 100 GA iterations with an initial population size of 200, averaged over 5 different seeds; baselines for number of pipe blocks, solvability and entropy fitness are indicated by dotted lines; shading indicates standard deviation.

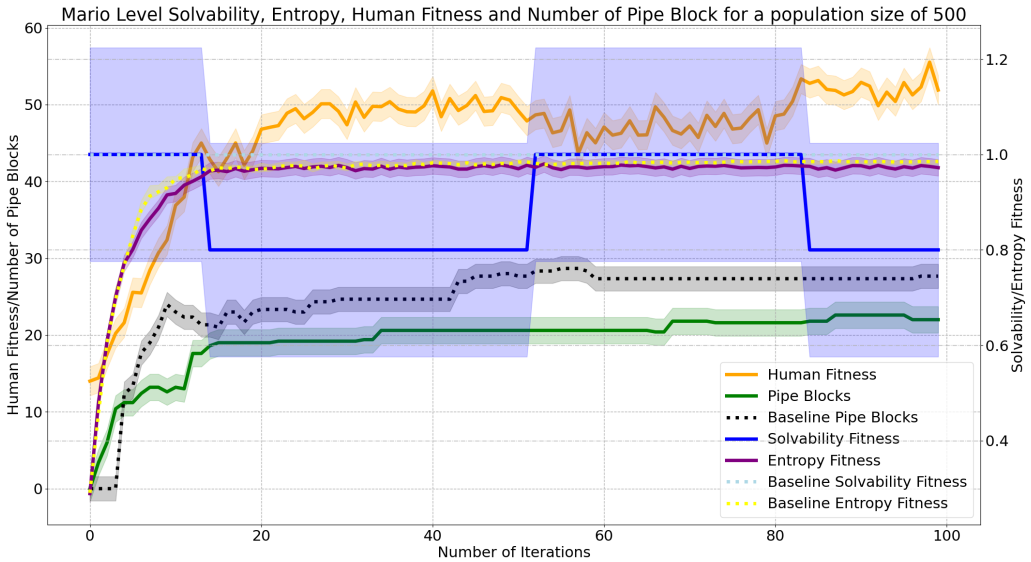


Figure 4.24: Figure showing solvability, entropy, human fitness and number of pipe blocks in the ground row of a *Mario* level for 100 GA iterations with an initial population size of 500, averaged over 5 different seeds; baselines for number of pipe blocks, solvability and entropy fitness are indicated by dotted lines; shading indicates standard deviation.

Chapter 5

Conclusion

The research presented a complete procedural content generation method in which a human's preference towards a 2D video game level's aesthetics can be used as a fitness function to a genetic algorithm in order to add a bias which causes future generated levels to contain more of the human's preferred features. This method successfully presents a way in which a numerical score can be assigned to a 2D video game level based on its aesthetics. By using TrueSkill and a round robin tournament, pairwise comparisons can be used to accurately rank each video game level based on a human preference of the levels' aesthetics. The level features can be extracted either using the binary level data itself, for *Maze* levels, or by using feature detection techniques such as SIFT, for *Mario*, and used to train an SVR model. This SVR model can accurately predict the TrueSkill scores for unseen video game levels and is passed into a genetic algorithm fitness function.

Experiments to better understand the proposed method have been conducted on two separate 2D platform video games, *Maze* and *Super Mario Bros*. There is a balance between the number of iterations and size of the initial population. For population sizes too small, the SVR does not perform well, whilst for population sizes too large diminishing returns are experienced. For the experiments conducted on *Maze*, the virtual human preference was black border pixels. *Maze* experienced an increase of 20% for the average number of black border pixels when human fitness is used showing the method is successful when a population size of at least 100 is used and a minimum of 100 iterations are run in order to ensure the levels are solvable. The experiments conducted on *Mario* used a visual human preference of pipe blocks in the bottom/ground row of the level. In *Mario*, feature vectors were extracted using SIFT, and the results show that using feature detection techniques is a viable option for extracting feature data from a 2D video game level to use in predicting a TrueSkill score. The results for *Mario* show that the proposed method can successfully be combined and optimised alongside other fitness functions, such as entropy and solvability. For initial population sizes larger than 200, there was no significant improvement on the number of pipe blocks but level solvability is negatively impacted, similarly to what was found in *Maze*. This lack of improvement indicates that the SVR model requires better training data,

rather than more training data to improve.

From the results obtained and discussed, it can be shown that human advice can successfully be incorporated as an objective in a multi-objective fitness function GA, using the implementation described in Chapter 3, which can be optimised as any normal fitness function would be. The major factors that need to be taken into account is the initial population size, which positively affects the virtual human preference but can negatively affect other game level characteristics, such as solvability. This relationship occurs due to the SVR model performing better with an increased amount of training data. In the case of *Mario*, the detail in the training data is more impactful for the SVR than the amount of training data. To cater for this, increasing the number of GA iterations allows for the virtual human preference to continually improve, whilst guaranteeing other game level characteristics are still adhered to. Multiple human inputs can potentially improve the proposed method, however further testing will be required before this can correctly be determined.

5.1 Future Work

Potential steps that can be followed in order to improve on the research discussed and results presented are listed in this chapter.

The use of feature detection techniques for video game levels is something that could be improved upon if multiple techniques were combined together to produce a large feature vector which could be fed into the SVR. This could potentially improve the performance of the SVR as well as the proposed methods generalisability for uses in other 2D platformer games.

We used virtual humans for the pairwise comparisons in order to provide a proof of concept. Future work would incorporate a real human study where they would provide better data on actual video game level design preferences. This data could be used in further research to make an impactful change to PCG and its uses in the game design field. A concern is using a round robin tournament style when humans are used as they would be required to perform a large amount of pairwise comparisons. An alternative method is to use a Swiss-system tournament style which would greatly reduce the number of pairwise comparisons required.

A grid search over the weights each of the fitness functions would be a good initial improvement in future work. This will ensure that the weights used will optimise the performance of the PCG implementation to improve on the human preference on levels generated, whilst adhering to the standard gameplay characteristics such as solvability. Alternatively, mutliobjective GA's could be used in place of the weighted sum approach

that was taken.

There are ways in which to optimise the PCG loop for multiple human advice instances, such as using a larger population time and iteration count. This process exponentially increases the processing time required, however if the time is available it could potentially produce large improvements over the results shown in this research.

We previously stated that procedural content generation in video games shows a lot of promise due to its ability to generate content automatically. So far, PCG has not fully delivered on this promise, as only characteristics such as solvabilty and difficult can reliably be implemented. We hope the findings of this research is a step towards improving PCG for video games by providing a way in which aesthetically pleasing features can be favoured in the generation of future video game levels.

References

[Barriga 2018] Nicolas A. Barriga. *A Short Introduction to Procedural Content Generation Algorithms for Videogames*, 05 2018.

[Bertozzi *et al.* 2007] M. Bertozzi, A. Broggi, M. Del Rose, M. Felisa, A. Rakotomamonjy, and F. Suard. A pedestrian detector using histograms of oriented gradients and a support vector machine classifier. In *2007 IEEE Intelligent Transportation Systems Conference*, pages 143–148, 2007.

[Beukman *et al.* 2022a] Michael Beukman, Christopher W Cleghorn, and Steven James. Procedural content generation using neuroevolution and novelty search for diverse video game levels. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '22, page 1028–1037, New York, NY, USA, 2022. Association for Computing Machinery.

[Beukman *et al.* 2022b] Michael Beukman, Steven James, and Christopher W. Cleghorn. Towards objective metrics for procedurally generated video game levels. *CoRR*, abs/2201.10334, 2022.

[Burke 2016] Michael Burke. Image ranking in video sequences using pairwise image comparisons and temporal smoothing. In *2016 Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech)*, pages 1–6, 2016.

[Chang and Lin 2002] Chih-Chung Chang and Chih-Jen Lin. Training v-Support Vector Regression: Theory and Algorithms. *Neural Computation*, 14(8):1959–1977, 08 2002.

[Ferreira *et al.* 2014] Lucas Ferreira, Leonardo Pereira, and Claudio Toledo. A multi-population genetic algorithm for procedural generation of levels for platform games. pages 45–46, 07 2014.

[Gomes *et al.* 2015] Jorge Gomes, Pedro Mariano, and Anders Christensen. Devising effective novelty search algorithms: A comprehensive empirical study. pages 943–950, 01 2015.

[Gravina *et al.* 2019] Daniele Gravina, Ahmed Khalifa, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis. Procedural content generation through quality diversity. In *2019 IEEE Conference on Games (CoG)*. IEEE, aug 2019.

[Hart *et al.* 1968] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[Herbrich *et al.* 2007] Ralf Herbrich, Tom Minka, and Thore Graepel. Trueskill(tm): A bayesian skill rating system. In *Advances in Neural Information Processing Systems 20*, pages 569–576. MIT Press, January 2007.

[Hochstrate *et al.* 2007] Nicola Hochstrate, Boris Naujoks, and Michael Emmerich. Sms-emoa: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181:1653–1669, 02 2007.

[Hsu *et al.* 2003] Chih-wei Hsu, Chih-chung Chang, and Chih-Jen Lin. A practical guide to support vector classification chih-wei hsu, chih-chung chang, and chih-jen lin. 11 2003.

[Jennings-Teats *et al.* 2010] Martin Jennings-Teats, Gillian Smith, and Noah Wardrip-Fruin. Polymorph: A model for dynamic level generation. 01 2010.

[Katoch *et al.* 2021] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. A review on genetic algorithm: Past, present, and future. *Multimedia Tools Appl.*, 80(5):8091–8126, feb 2021.

[Khalifa *et al.* 2020] Ahmed Khalifa, Philip Bontrager, Sam Earle, and Julian Togelius. Pcgrl: Procedural content generation via reinforcement learning. In *Proceedings of the Sixteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'20. AAAI Press, 2020.

[Linden *et al.* 2013] Roland Linden, R. Lopes, and Rafael Bidarra. Designing procedurally generated levels. pages 41–47, 10 2013.

[Liu *et al.* 2013] Jing Liu, Quan Wang, Chin yew Lin, and Hsiao wuen Hon. *Question Difficulty Estimation in Community Question Answering Services*, 2013.

[Mai Thanh Nhat Truong 2016] Sanghoon Kim Mai Thanh Nhat Truong. A review on image feature detection and description. *Proceedings of the Korea Information Processing Society Conference*, pages 677–680, 2016.

[Mawhorter and Mateas 2010] Peter Mawhorter and Michael Mateas. Procedural level generation using occupancy-regulated extension. pages 351–358, 08 2010.

[Minka *et al.* 2018] Tom Minka, Ryan Cleven, and Yordan Zaykov. *TrueSkill 2: An improved Bayesian skill rating system*. Technical Report MSR-TR-2018-8, Microsoft, March 2018.

[Najwa Mohd Rizal *et al.* 2022] Nur Najwa Mohd Rizal, Gasim Hayder, Mohammed Mnzool, Bushra M. E. Elnaim, Adil Omer Yousif Mohammed, and Manal M. Khayyat. Comparison between regression models, support vector machine (svm), and artificial neural network (ann) in river water quality prediction. *Processes*, 10(8), 2022.

[Ojala *et al.* 2002] Timo Ojala, Matti Pietikäinen, and Topi Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24:971–987, 2002.

[Pedersen *et al.* 2009] Chris Pedersen, Julian Togelius, and Georgios Yannakakis. Modeling player experience in super mario bros. pages 132 – 139, 10 2009.

[Raskar *et al.* 2015] Ramesh Raskar, Nikhil Naik, Jade Philipoom, and Cesar Hidalgo. Streetscore – predicting the perceived safety of one million streetscapes. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 01 2015.

[Scholkopf *et al.* 2000] B Scholkopf, Alex Smola, Robert Williamson, and Peter Bartlett. New support vector algorithms. *Neural computation*, 12:1207–45, 06 2000.

[Shannon 1948] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[Shu *et al.* 2021a] Tianye Shu, Jialin Liu, and Georgios N. Yannakakis. *Experience-Driven PCG via Reinforcement Learning: A Super Mario Bros Study*, 2021.

[Shu *et al.* 2021b] Tianye Shu, Jialin Liu, and Georgios N. Yannakakis. Experience-driven pcg via reinforcement learning: A super mario bros study. In *2021 IEEE Conference on Games (CoG)*, page 1–9. IEEE Press, 2021.

[Smola and Schölkopf 2004] Alex Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14:199–222, 08 2004.

[Sorenson and Pasquier 2010] Nathan Sorenson and Philippe Pasquier. Towards a generic framework for automated video game level creation. volume 6024, pages 131–140, 04 2010.

[Summerville *et al.* 2017] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. Procedural content generation via machine learning (pcgml). *IEEE Transactions on Games*, PP, 02 2017.

[Togelius *et al.* 2010] Julian Togelius, Mike Preuss, Nicola Beume, Simon Wessing, Johan Hagelbäck, and Georgios N. Yannakakis. Multiobjective exploration of the starcraft map space. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pages 265–272, 2010.

[Togelius *et al.* 2011] Julian Togelius, Georgios Yannakakis, Kenneth Stanley, and Cameron Browne. Search-based procedural content generation: A taxonomy and survey. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3:172 – 186, 10 2011.

[Vapnik 1995] V. Vapnik. *The Nature of Statistical Learning Theory*. 1995.

[Volz *et al.* 2018] Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M. Lucas, Adam Smith, and Sebastian Risi. *Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network*, 2018.

[Xiao *et al.* 2010] Jianxiong Xiao, James Hays, Krista Ehinger, Aude Oliva, and Antonio Torralba. Sun database: Large-scale scene recognition from abbey to zoo. pages 3485–3492, 06 2010.

[Yannakakis and Togelius 2011] Georgios N. Yannakakis and Julian Togelius. Experience-driven procedural content generation. *IEEE Transactions on Affective Computing*, 2(3):147–161, 2011.