Integrating Player-Centric Procedural Content Generation in a Human Testing Environment

Jaran I. Holt
University of Utah

UUCS-25-002

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

16 April 2025

Abstract

I present an attempt to integrate artificial intelligence learning systems into video game procedural content generation to adapt to a player's interests dynamically. Artificial intelligence has developed rapidly over the past few years, and the possibility of applying it in entertainment environments is yet to be explored in an engaging yet ethical manner. Procedural Content Generation (PCG) is already used to help create unique levels or change the difficulty without needing explicit information. By integrating support vector machines (SVM) into a procedural content generator, player-centric procedural content generation (PCPCG) can create unique and personalized experiences for the players. Based on on-the-fly, sparse data collection, PCPCG learns from players themselves as they play through different levels to generate future content, allowing for following playthroughs to match player preferences. This research attempts to analyze the effectiveness of PCPCG in a Pac-Man game environment when humans play through.

Previous work on PCPCG in video games has used computer-generated players with different preferences for what would most engage them in their environment. As a part of this new research, an experiment was conducted involving two groups playing a recreation of Pac-Man and recording their preferences. One group played through 30 minutes of levels that would randomly generate. Another group would play through 30 minutes of levels, but would have the environments generated using PCPCG with multiple SVMs recording. It was observed that between the control and test groups, the difference in terms of enjoyment was positive but statistically insignificant. Many players saw potential in the system if further refined and implemented into other genres.

INTEGRATING PLAYER-CENTRIC PROCEDURAL CONTENT GENERATION IN A HUMAN TESTING ENVIRONMENT

by

Jaran I. Holt

A Senior Honors Thesis Submitted to the Faculty of
The University of Utah
In Partial Fulfillment of the Requirements for the

Honors Degree in Bachelor of Science

In

Computer Science

Approved:

Daniel S. Brown Faculty Thesis Mentor	Mary Hall Chair, School of Computing
Thomas Henderson Departmental Honors Liaison	Monisha Pasupathi, PhD Dean, Honors College

May 2025
Copyright © 2025
All Rights Reserved

ABSTRACT

I present an attempt to integrate artificial intelligence learning systems into video game procedural content generation to adapt to a player's interests dynamically. Artificial intelligence has developed rapidly over the past few years, and the possibility of applying it in entertainment environments is yet to be explored in an engaging yet ethical manner. Procedural Content Generation (PCG) is already used to help create unique levels or change the difficulty without needing explicit information. By integrating support vector machines (SVM) into a procedural content generator, player-centric procedural content generation (PCPCG) can create unique and personalized experiences for the players. Based on on-the-fly, sparse data collection, PCPCG learns from players themselves as they play through different levels to generate future content, allowing for following playthroughs to match player preferences. This research attempts to analyze the effectiveness of PCPCG in a Pac-Man game environment when humans play through.

Previous work on PCPCG in video games has used computer-generated players with different preferences for what would most engage them in their environment. As a part of this new research, an experiment was conducted involving two groups playing a recreation of Pac-Man and recording their preferences. One group played through 30 minutes of levels that would randomly generate. Another group would play through 30 minutes of levels, but would have the environments generated using PCPCG with multiple SVMs recording. It was observed that between the control and test groups, the difference in terms of enjoyment was positive but statistically insignificant. Many players saw potential in the system if further refined and implemented into other genres.

For my parents, Carson and Marlene Holt

TABLE OF CONTENTS

ABSTRACT ii

LIST OF FIGURES v

1 INTRODUCTION 1

2 BACKGROUND AND RELATED WORK 3

3 DEFINING PCPCG 6

3.1 SVMs and Features 6

3.2 Collecting Feedback 7

4 IMPLEMENTATION 9

4.1 Implementation of PCPCG 9

4.2 Level Generation 9

4.3 Information Collection 10

5 METHODS 11

5.1 Possibility Space 11

5.2 Testing PCPCG 13

6 PARTICIPANT RESPONSE 17

6.1 Early Experience 17

6.2 Late Experience 18

7 RESULTS 20

8 FUTURE WORK 21

9 CONCLUSION 22

REFERENCES 23

LIST OF FIGURES

Algorithm 1: Support Vector Machines with PCPCG 7

Figure 1: Rejection sampling visualized 8

Figure 2: The original Pac-Man maps and a Generation 12

Figure 3: PCPCG-Generated Layouts 12

Figure 4: Tracking the average preferences of players per group. 14

Figure 5: Average enjoyment per category for each group. 14

Figure 6: Average enjoyment as a whole for each group. 15

Table 1: T-Distribution and P-Value of the data taken as a whole 15

Table 2: P-Value per Bin 16

1 INTRODUCTION

Video games are known for having many complex decisions presented that must be made at a moment's notice. This creates an excellent environment to train artificial intelligence in more dynamic ways. One way AI is implemented into games is in the development cycle to assist human developers in creation, whether offline or online. Procedural content generation (PCG) involves the dynamic creation of content with minimal human feedback to bring about longer game experiences than with usual game design [7], though it typically uses a different kind of AI than is used in academics.

Some games also contain dynamic game balancing (DGB), where the game will adjust how easy or hard it is based on the player's experience when playing, helping to ensure that the games are challenging but not frustrating [3]. This can be more personal and enjoyable than regular difficulty, learning dynamically as the user plays, but the content itself stays static [17, 20].

PCG and DGB often fall into one of two categories: *Offline*, where human refinement is done before distribution, and *Online*, where adaptation is done during runtime. The latter is most often reserved for DGB. The training of AI in complex environments and the dynamic creation of levels form the foundation of *player-centric procedural content generation* (PCPCG). By gathering feedback from the player as the game progresses, PCPCG can use that feedback to dynamically generate new content. By learning from past level playthroughs and by using small surveys to understand level preferences, new levels can be made that fit into the changing interests of the player.

I present my efforts to test a simple PCPCG model in a human environment. Since humans have more diverse and unquantifiable interests, my goal was to try to track

different variables using multiple support vector machines to account for different playstyles. The game receives feedback at run time and applies preference learning to a runtime-adaptable PCG. As a part of my research, I compare the results of this with the results of testing another group of people playing with purely random generation. While predicting that PCPCG would learn from player preferences to create levels they enjoy more, whether they play for score or easier levels, the results show minimal improvement compared to random generation.

I present the methodology, results, and findings, and discuss limitations in my study, as well as what the future could hold for player-centric procedural content generation. Using a Pac-Man clone environment designed for Unity [14], I implemented a version of PCPCG that uses multiple SVMs to track different preferences, including the pellet arrangement, the density of power pellets, level generation in-game, and the frequency at which fruits spawn in the game. I then tested how players engaged with the game compared to a purely random version of the game by using 10-point Likert scales at the end of each level to ask about their preferences. Additionally, I had all participants fill out surveys at different points as they played the game to gather feedback about how it could be improved and what they enjoyed about their experience.

While the results didn't show with full certainty that PCPCG would work in games as it currently is, it does trend towards being something that could be built upon, with many of the players opining that PCPCG definitely could be useful if fine-tuned. Making the questions more subtle or having the answers be on a wider range than just a simple "like and dislike" binary scale are areas to look into improving PCPCG for the future, with more playtesting done to help reduce player boredom.

2 BACKGROUND AND RELATED WORK

Procedural content generation is the ability for a game to create content with minimal human intervention on the developer's side. These can include the generation of maps based on a numerical seed or new obstacles being generated by the buttons a player presses. [7] Because of it having little developer involvement, there are many ways to implement it.

Procedural content generation via machine learning (PCGML) has attempted to help tune how PCG already works [6]. It can work well with a human designer who provides example data, which the program can then use to make levels that are beatable, yet challenging. This helps not only reduce time and costs, but also reduces storage while helping raise entertainment value. Experiments have been done with Super Mario Bros.-styled environments using Markov chains and a computer player to make sure the levels are still beatable.

These kinds of methods often use patterns of the level layout as factors in a weighted matrix. These simpler representations can be used to even generate new environments [6]. However, data can end up being very sparse due to the limitations of how many different games within the genre are available to experiment with. Online corpora have been made to try to resolve these issues, combining different level formats to allow for the large data sets typically needed [13].

Reinforcement learning has been used not only on-the-fly, with the reward being given as the program goes on, but used offline when setting up challenges for the players. These have all been used for a variety of implementations such as level recommendation, match making, and computer players who play like people [6, 9, 10, 12]. Reinforcement

learning, however, requires a lot of data. For something that requires sparse data, it is better to use a support vector machine, or an SVM [19, 21]. Similar to reinforcement learning, SVMs can be run on the fly. By passing in various "support vectors" labeled positively or negatively, the SVM will attempt to divide out the labeled items and maximize the margin between them in a given hyperplane, or higher-dimensional space. This works well if with human feedback based on a general positive or negative rating [18].

As experiments with content generation often focus on platformer environments, Infinite Mario Bros. is commonly used to test out new methods [22]. In the base environment, a player is given a basic level to run through and, based on how they interact, a new level is made the next time they play. While not like other procedural content generation systems, opting to use button inputs rather than seeds for generation, Infinite Mario Bros. continues to act as a major inspiration for similar projects, such as using LLMs to input a desired level and having the output be generated by user request [4].

Player-centric Procedural Content Generation (PCPCG) was initially proposed by N. Blackburn and M. Gardone under the guidance of D. Brown of the University of Utah [1]. The purpose of PCPCG was to allow for online content generation seen in past experiments, but to use the player's preferences as they play the game to make new levels based on changing needs. In the original workshop paper, computer-generated players were designed with varying "preferences" that detailed what they wished to get out of the game. A simple Pac-Man environment was used where, based on the preferences, different numbers of pellets in different locations would appear. The model would pin

down what the models were interested in and attempt to adjust when there was a spontaneous change in player preference. Preferences for each player were determined by their "personalities," influencing what they rated positively and negatively, while the support vector machine (SVM) behind the PCPCG adapted in real time. My work builds on this by performing human trials, letting humans respond about their enjoyment of the game on a 0-to-9 scale, with multiple different SVMs being used to calculate their enjoyment.

3 DEFINING PCPCG

For the PCPCG model to work in a human testing environment, the algorithm must be able to operate with minimal training, as longer training in terms of both play sessions and determining playing interests both run the risk of the player becoming disinterested in the game itself. They need to be online learners, gathering information at runtime to change based on whoever is playing rather than off of any predetermined assumptions. Since data is obtained infrequently, the algorithm must be able to work even given sparse feedback. This must apply even given a higher number of features.

3.1 SVMs and Features

Support Vector Machines (SVMs) have the dimensionality of their hyperplane determined by the number of features and the dimensionality needed to make them linearly separable [21]. Starting empty, the SVM will gather more data points, each labeled and placed somewhere on the hyperspace. With this data, the SVM will create an equation that separates the points via a hyperplane while maximizing the margin between different points [18]. This can be linear, but different kernels can be applied to change the shape of the hyperplane [19]. For PCPCG, the point's label is determined by the player, being positive or negative based on their own experience with the given features.

By tracking our current preferences and their labelling, the SVM can guess future preference labels and select one it believes will be positive. Following Algorithm 1, when a new point is added and labeled, the weight matrix defining the SVM is changed to label the points correctly while maximizing the margin between the points and the hyperplane. Once the player has labeled the newly presented feature, then the given label, together with the old labels, can be used to retrain the SVM to generate better levels for the future.

Algorithm 1: Support Vector Machine with PCPCG

Data: svmDivide: weights that determine the boundary of an SVM, prevData: Labeled data already present in the model, oldPref: the prior preference used, rating: the user's experience on the level, ss: step size for next preference
Result: newPref

1. prefs ← (oldPref, rating)
2. newPref = null
3. for $p \epsilon$ oldPref do:
    a. prefVal ← GetNextPreference(*p,*, ss)
    b. while prefVal * svmDivide < 0 do
        i. prefVal ← GetNextPreference(*p,*, ss)
    c. end
    d. newPref.append(p.name, prefVal);
4. prefs.append(newPref)
5. update(svmDivide, prefs)
6. end

There are different parts of a game that a player may like as a result of their play style. A collection of SVMs can be used to define different feature sets based on the roles they fill. Each of these will learn the player's preferences separately and pass the results into the main program to generate content.

3.2 Collecting Feedback

Feedback requests only happen at natural breaks in the game, primarily at the end of the level. A 10-point Likert scale, used to ask for the player's opinion on a feature on a range from 0 to 9, is presented for each set of features that must be ranked. This helps quantify player enjoyment for future levels and for recording data to analyze. The process is very noisy, so the initial levels generated are done with random values to gauge the player's preferences first. Because the data is very noisy, a static list structure was used to gather as much data as possible.

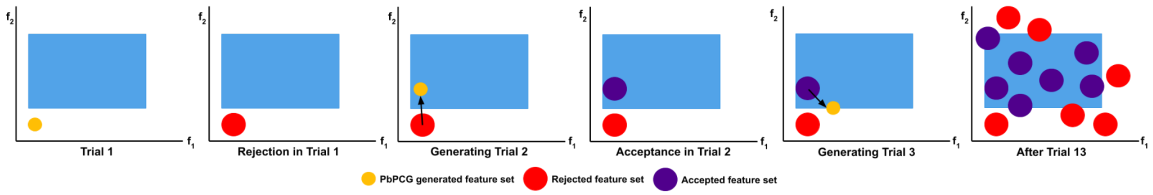

Figure 1: Rejection sampling visualized. An illustration of PCPCG learning player preferences using rejection sampling and the previous sample to generate the next one. The blue region marks the player's preferences, with every point outside of it being rejected and marked in red. Purple circles indicate an accepted sample.

Rejection-based sampling is used to learn player preferences. An ϵ-ball (epsilon ball) centered on the feature vector is created. Upon completion of the level, the player determines how much they liked it, either accepting (liking) it or rejecting (disliking). This label is then applied to the ϵ-ball, becoming a dead zone if rejected, preventing further feature vectors from being created there (illustrated in Figure 1). If the player accepts the feature, future features can be made within the ϵ-ball region. New feature generation follows Algorithm 1, taking a predetermined number of steps to a new feature vector.

4 IMPLEMENTATION

This section explains how the player-centric procedural content generation (PCPCG) is implemented, the game used for the implementation, and changes made to the game to make the experience more dynamic.

4.1 Implementation of PCPCG

The game used is a modified version of Zigurous's open-source Pac-Man clone for Unity [14], providing basic art and assets to recreate the Pac-Man game. This was modified to allow for unique maps to be created after each iteration of gameplay [15]. Additionally, fruit spawning was implemented, with fruits in Pac-Man rewarding a different amount of points based on the kind that spawns on the map. Unlike the original game, the fruits spawn on the map randomly with a probability-per-pellet eaten as determined by the PCPCG. The main generation algorithm was added from here, using multiple SVM samples to focus on different features in the feature space. A total of four samples were used. The preference learning agent would create dead zones on content that was disliked, given a 0 on the Likert scale, adding constraints for future levels. All other levels were labeled on a scale from -1 to 1, including decimals, based on the ranking given by the player. The recorded features would be used as a reference for generating new levels. Within the constraints provided, a new single level is randomly generated and presented to the player.

4.2 Level Generation

As one feature I wanted to test was how players reacted to different levels, I decided to use Shaun LeBron's Pac-Man maze generation code [15], which uses a method of placing Tetris blocks in an invisible environment and changing the width and

height of blocks create gaps in between that act as the maze path that Pac-Man and the ghosts can move on. The code itself was written for JavaScript, so I had to port it to C# to be compatible with the game.

I added more parameters to the cells that make up the blocks placed down, explicitly defined the variables present, and had the output be done as a text file. This file would then be mapped to Unity's tile map editor, with the various turns in the maze using invisible node tiles to tell the ghosts the available directions. To work with PCPCG, I took the elements of code that determined the probability of blocks of certain sizes being placed and the maximum number of long blocks that would be used. These would act be changed by features of one of the SVMs to predict more desirable levels..

4.3 Information Collection

The controller for PCPCG has a built-in method that stops gameplay when called to pull up a question about the gameplay. While this prompt is up, the player can press a number from 0 to 9 to answer. This number is then converted to a different value ranging from -1 to 1, which would then be passed into an SVM based on the question asked. This sample would tell the SVM the label of the current feature for it to compute a new feature for the next level.

I added a path variable that leads to Unity's "persistent data path," a location in storage for games to place their save data to allow for saving and loading. I used this and C#'s general IO library to create a text document and write the results of the player's responses as they played through. Once the player was done, I would have them press a designated button to call Unity's debug log, which would read and print this data so I could record it.

5 METHODS

As these trials are done by humans, I needed a way to quantify their level of enjoyment to measure how well the PCPCG was working. To do this, I selected different attributes of the game and had them be modified as the game progressed.

5.1 Possibility Space

To accommodate different play styles, four general categories were recorded to manipulate in future levels. Each of these categories had its own set of features.

(1) Pellet Layout: Taken from the original experiment [1], the placement of pellets and how many exist on the world map is controlled.

- Total Pellet Count: ($C_{tp} = rp + pp$) Total pellets (tp) on the map. The value is at least 1, as 0 pellets act as the win condition. Range changes depending on map layout.
- Total Pellet Density: ($D_{tp} = \frac{tp}{s}$, Range: (0:1] ϵ $\mathbb{R}$) Ratio of total pellets to all pellet tiles (s).
- Symmetry/Asymmetry (S, Range: [0, 1] ϵ $\mathbb{Z}$): If true, would draw pellets symmetrically along the vertical axis, using the same symmetry as the map generation. This stayed random.

(2) Power Pellet Density: ($D_{pp} = \frac{pp}{tp}$, Range: [0, 1] ϵ $\mathbb{R}$): Ratio of power pellets (pp) to total pellets.

(3) Map Layout: Using LeBron's maze generation code [15], different parameters involved in the probability of the maze layout generating in different patterns were manipulated. This includes the probability to stop pieces from growing at smaller cell

sizes of 2, 3, or 4; the probability of top and bottom cells joining; the probability of cells extending at sizes 2 or 4; and the number of long piece cells allowed in the map for longer hallways in the maze. Each of these parameters had a range of [0.01, 0.75]. The long piece cell count has a range of [0,4].

(4) Fruit Spawn Rate: (F, Range: $[0, 0.075] \in \mathbb{R}$). The likelihood of a fruit spawning at a random place on the map after eating a pellet.

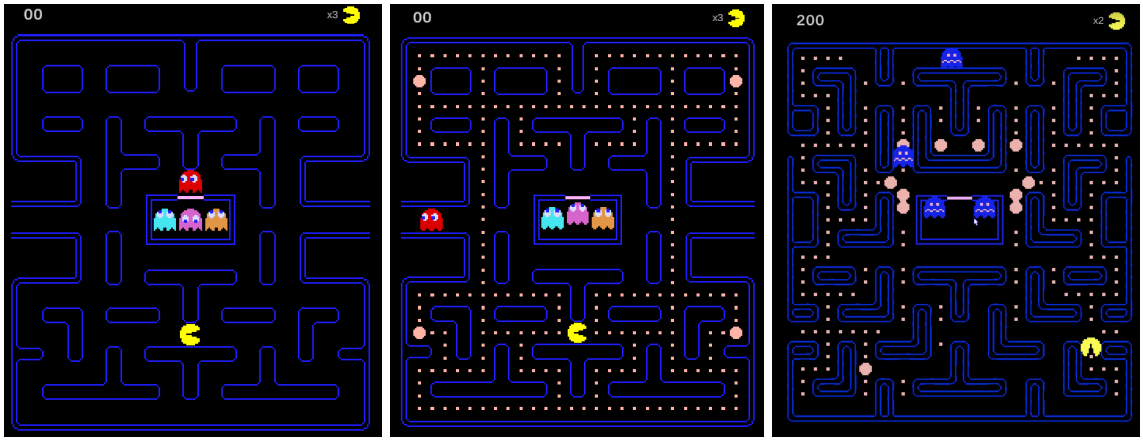

Figure 2: The original Pac-Man maps and a Generation. Shown without (left) and with (center) pellets vs. a PCPCG-generated map (right). (Pac-Man art assets are provided by Zigurous.)

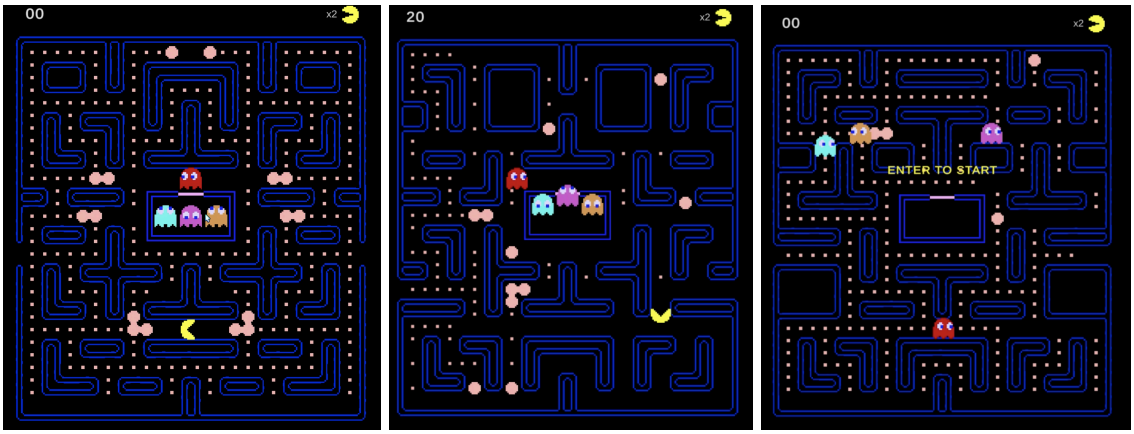

Figure 3: PCPCG-generated layouts. Show difference in power pellet density, pellet count, and map layout. (Pac-Man art assets are provided by Zigurous.)

5.2 Testing PCPCG

To test the effectiveness of PCPCG on human players, trials were conducted in which people were split into two groups, with both of them playing a single level of standard Pac-Man to familiarize themselves with the game [16]. Doing this helped to ensure every participant was familiar with the basics of how Pac-Man plays.

The control group played a version of Pac-Man where all of the possible parameters were picked by random number generators throughout, no matter what rating they gave on the 10-point Likert scale. The main testing group played the version of Pac-Man with PCPCG implemented. After each level, the parameters used would generate a new map, Figure 2 showing a basic example. This could result in several different outputs, all seen in Figure 3. Both groups would play their game for 30 minutes in total, answering the Likert questions on a scale from 0-to-9 and playing through the levels at their own pace. In addition, a survey was conducted every 15 minutes to evaluate each player's opinions on the game as they played, as well as their thoughts on PCPCG as a concept. Data from playtesting was saved in a text file as they answered questions at the end of each level.

Once the playtesting was done, the data was extracted and split between the control group and the test group. The data was averaged out between 6 bins for each player, a bin representing 5 minutes of gameplay. This data was then averaged out to generate a score graph for each sample between the two groups. The score graphs could then be compared, using standard deviation to determine the error range and t-testing to determine the likelihood of these results happening by chance.

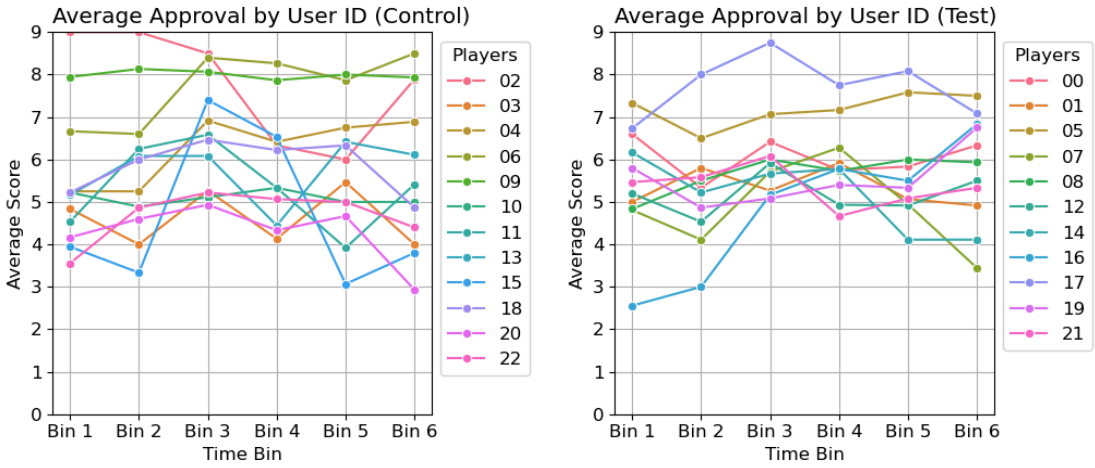

Figure 4: Tracking the average preferences of players per group. Each player's gameplay was split into 6 different bins, representing 5 minutes of gameplay. The approval was determined by averaging each player's score for pellet layout, power pellet density, map layout, and fruit frequency.

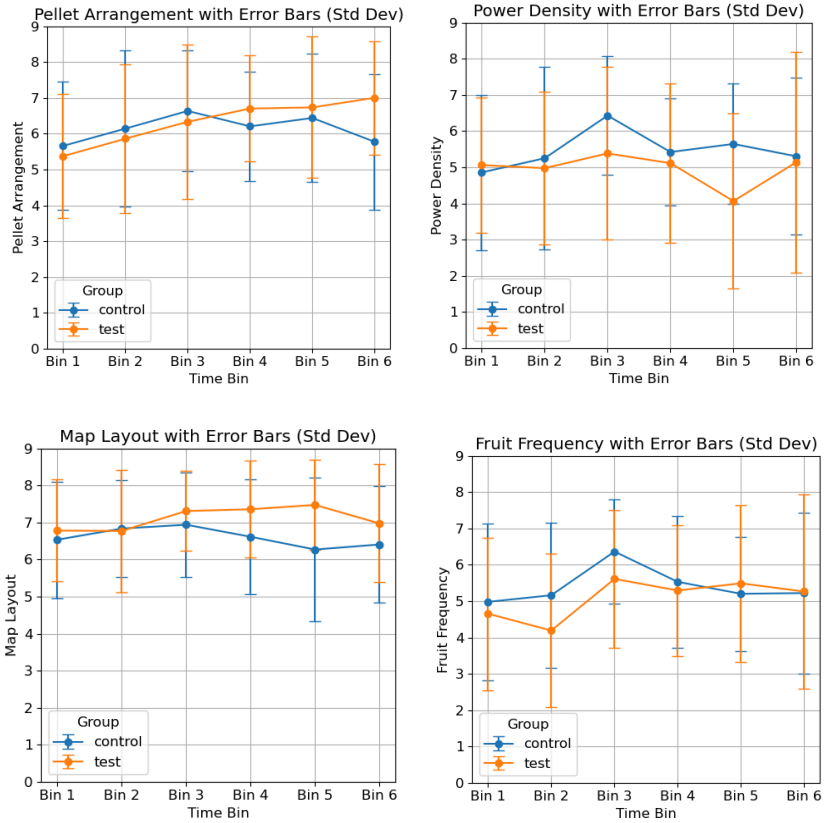

Figure 5: Average enjoyment per category for each group. Shown in the graphs are pellet arrangement (top-left), power density (top-right), map layout (bottom-left), and fruit frequency (bottom-right). The control group played Pac-Man with pure random generation, while the Test group played with PCPCG implementation.

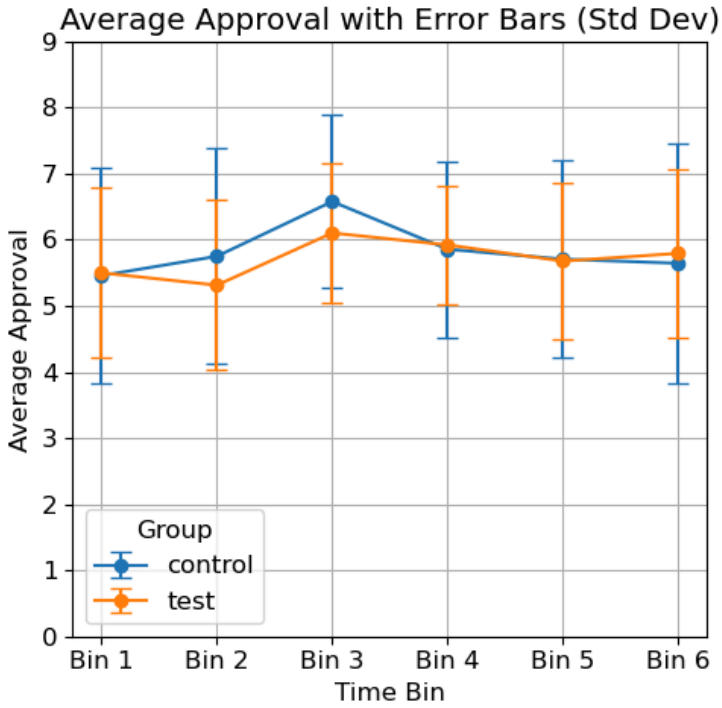

Figure 6: Average enjoyment as a whole for each group. The control group played Pac-Man with pure random generation, while the Test group played with PCPCG implementation.

Table 1: T-Distribution and P-Value of the data taken as a whole. Underlined is data with a P-value ≤ 0.05.

Feature	*Pellet Layout*	*Power Density*	*Map Layout*	*Fruit*	*Average*
T-Distribution	-0.6133	1.4363	-2.0749	0.9548	0.3415
P-Value	0.5407	0.1534	**0.0399**	0.3415	0.6197

Table 2: P-value of the data per bin. Underlined is data with a P-value ≤ 0.1.

	Pellet Layout	*Power Density*	*Map Layout*	*Fruit*	*Average*
Bin 1	0.7005	0.8107	0.6931	0.7173	0.9470
Bin 2	0.7534	0.7803	0.9189	0.2712	0.4827
Bin 3	0.7131	0.2417	0.4856	0.2985	0.3440
Bin 4	0.4357	0.7020	0.2269	0.7553	0.8898
Bin 5	0.7127	**0.0881**	**0.0883**	0.7207	0.9588
Bin 6	0.1060	0.8853	0.3991	0.9671	0.8205

Results of p-value evaluation are shown in Tables 1 and 2, showing how likely these could have happened by chance. Graphs shown in Figure 5 show the results of the average enjoyment of features per group, with standard deviation to show possible error. Figure 6 shows the average enjoyment per level after averaging the features together. PCPCG with more accuracy on Map Layout and Power Density, especially later in the game. Both tables give different results, suggesting PCPCG is good at map layout but not good at finding the density of power pellets needed.

6 PARTICIPANT RESPONSE

The participants involved were random volunteers told that they'd be playing Pac-Man with procedural content generation. They would answer survey questions in addition to what was presented at the end of each level every 15 minutes of playing. The surveys asked about initial expectations, experience both early on and later, anything they found particularly interesting, and what their thoughts were on PCPCG as a concept. This helps gauge how effective the algorithm is and what changes could be made for it in future experiments. While most of the players were familiar with Pac-Man, some hadn't played it in a long time or had played it for their first time through the experiment. Some were unfamiliar with game design, while others were studying it. This helped provide a wide range of personalities for the feedback.

6.1 Early Experience

Many players were generally surprised by the game early on, seeing the different maps. For some players, they felt it was a nice change of pace from the Pac-Man they were used to, finding the map layouts to be difficult to adjust to. The initial points were described as challenging by players more inexperienced with Pac-Man, with participant 9 noticing that while he found the game easier, he could tell the ghosts were faster and more aggressive. Some of the players remarked that it "was interesting to see the different combinations of level layouts, particularly regarding power pellets, as these kinds of games tend to have few pellets." Participants generally had high expectations, knowing that they would be playing Pac-Man with some degree of generation in-game.

6.2 Late Experience

Players were generally accustomed to the game after the first half of the experiment. At this point, most players found the game easy, mostly because of the power pellet density. As pellets tended to be placed close together, it made it easy to stay powered-up when eating the ghosts. Many players took issue with this, saying they would like it changed since having too many pellets "neuters the potency of the Ghosts." This was the point where preferences in gameplay style were made clear. For two players in the control group, they "had objectives change as they played to maximize their score." This is contrasted with one player who said the abundance of pellets made him feel that score wasn't worth going for. Most of the players who felt the levels were similar were those in the control group, such as participants 16 and 20. Participant 20 was mainly encourage to go for score as a result. Participant 16 was similar, saying that level generation "seemed to be less accurate" later on.

Map layout was a more divisive system. Several players early on expressed shock at seeing how different the levels were. As they played, they would split in terms of opinion. Participant 14 believed that the map layouts were more challenging, with the frequency of more engaging levels growing over time. Participant 23 shared this opinion, saying that "the levels were consistently high quality throughout." In contrast, participant 9 believed the levels didn't matter if he could keep collecting pellets.

Players all reported on their "attention grabbers," something that caught their eye as they played. The answers to this varied, with some players praising the ability for the maps to change and be something the game learns from. Participant 5 was one such player, finding enjoyment in making a plan given a map layout before starting. Some

were more negative, doubling down on the issue of power pellets sometimes being placed in a row and making the difficulty of the game too easy. Despite this, when asked about what they thought of PCPCG as content, they continued to believe that it had a lot of potential. Participant 11 did feel that a Likert scale wasn't the best way to gather information on players unless they could somehow be very specific about questions, and Participant 13 felt the data needed to be collected more subtly, but they both felt nonetheless that games that are simpler in design could use PCPCG to personalize player experience.

7 RESULTS

The results in Table 1 suggest that a lot of the data could have happened by pure chance, meaning the results are generally irrelevant. However, the p-value for power density shows that there's an 85% chance of repeating these result, and the p-value for the map layout shows that there is a 95% chance. Looking closely at the p-values per time frame, they suggest that as the gameplay continues for longer, the repeatability for pellet layout, power density, and map layout generally goes up, meaning that more play time yields more measurable results.

Looking at the results with the lowest p-value in each column, they tend to be at around the second half of the experiment, when the difference between intentional generation versus random is at its greatest. Comparing the graphs together in Figure 5, the tests generally show greater results in pellet layout and especially in map layout. Power density seems to be seen as worse generally, suggesting PCPCG did not improve people's experience here and may have hindered it. This goes along with the most common criticism of both groups, that being that the power pellets bunched up too often. Additionally, looking at the error bars in Figure 5, they suggest that it's possible that random results and PCPCG could have overlap, meaning that more testing needs to be done. In general, the improvement in player experience provided by PCPCG was on average minimal, as Figure 6 shows that while PCPCG did better than random generation during the second half of the experiment, randomness did better early on. This is further exemplified by the error bars for the PCPCG group being almost fully enveloped by the ones in the control group.

8 FUTURE WORK

The current implementation of PCPCG is limited in what it can do. One limitation it currently has is with receiving feedback from humans. When accepting a broader range of feedback, the algorithm fails to properly label a result as positive or negative unless it is exactly +1 or -1. Since it's possible for rankings to be in between these two, it means that unless the answer is always positive or always negative, the results of running PCPCG will be very close to random, just in a narrower range than pure randomness. Implementing a continuous value SVM into PCPCG could be helpful for future experiments.

Real-time player feedback is another issue PCPCG runs into. Constantly needing to provide feedback, especially as a survey, can be annoying and hinder player engagement, something I observed with some players repeatedly asking if they were done after answering some in-game questions. Being able to implement feedback more subtly with everyday interactions, such as a "retry" button for an enjoyable level, could help mitigate frustration caused by end-of-level questionnaires.

Having samples built on specific play styles as a form of pre-training could help players early on. Rather than having to fully train the algorithm on their data, they could choose a starting sample that suits specific play styles and proceed from there, helping give a head start in determining what the player finds active engagement in. Feature sets could explore previously disliked areas, using the probability of liking a level or having high enjoyment for specific gameplay parameters. Experiments done with these alterations could yield more promising results.

9 CONCLUSION

Player-Centric Procedural Content Generation is a way to implement player feedback to generate content, such as levels in games, to accommodate different play styles. Players who have played games with it do see potential in it and think it could help personalize gameplay experience, showing both the worth in further development and the importance of playtesting and human fine-tuning in making it functional. However, based on statistical analysis, PCPCG in its current state provides minimal improvements in gameplay experience over random results. PCPCG still needs more development before it can be fully implemented into game environments. Further work in regards to scalability and passive implementation is necessary to allow it to help improve player experience. Work using continuous SVMs or changing the method of confirming preferences using the sign of the numerical response is needed to see the full promise of PCPCG. Hiding the data collection to prevent interfering with gameplay and adding pre-training could help with future trials to make PCPCG more effective in the game world. By making these changes and running playtests, PCPCG offers personalized experiences that can greatly engage many differnet types of players.

REFERENCES

[1] N. N. Blackburn, M. Gardone, and D. S. Brown, “Player-Centric Procedural Content Generation: Enhancing Runtime Customization by Integrating Real-Time Player Feedback,” in Companion Proceedings of the Annual Symposium on Computer-Human Interaction in Play, Stratford ON Canada: ACM, Oct. 2023, pp. 10–16. doi: 10.1145/3573382.3616069. Available: https://dl.acm.org/doi/10.1145/3573382.3616069.

[2] D. S. Brown, R. Coleman, R. Srinivasan, and S. Niekum, “Safe Imitation Learning via Fast Bayesian Reward Inference from Preferences,” arXiv.org, Feb. 21, 2020. Available: https://arxiv.org/abs/2002.09089v4.

[3] M. Mario, A. Levina, and Y. Arifin, “A Review of the Trends, Methods, and Impacts of Dynamic Game Balancing,” International Journal of Computing and Digital Systems, vol. 16, no. 1, pp. 1–13, Aug. 2024, Available: https://journal.uob.edu.bh:443/handle/123456789/5865.

[4] L. van Koppen, “Integrating a Human Feedback Loop in PCG for Level Design Using LLMs,” Jul. 23, 2024. Available: http://essay.utwente.nl/102052/. [Accessed: Oct. 09, 2024]

[5] M. R. Johnson, “Playful Work and Laborious Play in Super Mario Maker,” Digital Culture & Society, vol. 5, no. 2, pp. 103–120, Dec. 2019, doi: 10.14361/dcs-2019-0207. Available: https://www.degruyter.com/document/doi/10.14361/dcs-2019-0207/html.

[6] A. Summerville et al., “Procedural Content Generation via Machine Learning (PCGML),” arXiv.org, Feb. 02, 2017. Available: https://arxiv.org/abs/1702.00539v3. [Accessed: Oct. 09, 2024]

[7] J. Togelius, E. Kastbjerg, D. Schedl, and G. N. Yannakakis, “What is procedural content generation?: Mario on the borderline,” in Proceedings of the 2nd International Workshop on Procedural Content Generation in Games, Bordeaux France: ACM, Jun. 2011, pp. 1–6. doi: 10.1145/2000919.2000922. Available: https://dl.acm.org/doi/10.1145/2000919.2000922.

[8] W. Westera et al., “Artificial intelligence moving serious gaming: Presenting reusable game AI components,” Educ Inf Technol, vol. 25, no. 1, pp. 351–380, Jan. 2020, doi: 10.1007/s10639-019-09968-2. Available: http://link.springer.com/10.1007/s10639-019-09968-2.

[9] D. Hind and C. Harvey, “Using a NEAT approach with curriculums for dynamic content generation in video games,” Pers Ubiquit Comput, vol. 28, no. 3–4, pp. 629–641, Aug. 2024, doi: 10.1007/s00779-024-01801-z. Available: https://link.springer.com/10.1007/s00779-024-01801-z.

[10] P. D. Paraschos and D. E. Koulouriotis, “Game Difficulty Adaptation and Experience Personalization: A Literature Review,” International Journal of Human–Computer Interaction, vol. 39, no. 1, pp. 1–22, Jan. 2023, doi: 10.1080/10447318.2021.2020008. Available: https://www.tandfonline.com/doi/full/10.1080/10447318.2021.2020008.

[11] H. Schaa and N. A. Barriga, “Evaluating the Expressive Range of Super Mario Bros Level Generators,” Algorithms, vol. 17, no. 7, p. 307, Jul. 2024, doi: 10.3390/a17070307. Available: https://www.mdpi.com/1999-4893/17/7/307.

[12] “Training an unbeatable AI in Trackmania - YouTube.” Available: https://www.youtube.com/watch?v=Dw3BZ6O_8LY.

[13] A. J. Summerville, S. Snodgrass, M. Mateas, and S. Ontañón, “The VGLC: The Video Game Level Corpus.” arXiv, Jul. 03, 2016. Available: http://arxiv.org/abs/1606.07487.

[14] Zigurous. 2021. Pac-Man (2D) [Source Code]. https://github.com/zigurous/unity- pacman-tutorial

[15] Shaun LeBron. 2012. Pac-Man Maze Generator [Source Code]. https://github.com/shaunlebron/pacman-mazegen

[16] Shaun Williams. 2022. Pac-Man (HTML) [Source Code]. https://github.com/masonicGIT/pacman

[17] Miguel González-Duque, Rasmus Berg Palm, and Sebastian Risi. 2021. Fast Game Content Adaptation Through Bayesian-based Player Modelling. arXiv:2105.08484 [cs, stat] http://arxiv.org/abs/2105.08484

[18] “What Is Support Vector Machine? | IBM,” Dec. 12, 2023. Available: https://www.ibm.com/think/topics/support-vector-machine.

[19] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt and B. Scholkopf, "Support vector machines," in IEEE Intelligent Systems and their Applications, vol. 13, no. 4, pp. 18-28, July-Aug. 1998, doi: 10.1109/5254.708428.

[20] Robin Hunicke and Vernell Chapman. 2004. AI for Dynamic Difficulty Adjustment in Games.

[21] W. S. Noble. What is a support vector machine?. Nat Biotechnol 24, 1565–1567. 2006. https://doi.org/10.1038/nbt1206-1565

[22] N. C. Hou, N. S. Hong, C. K. On and J. Teo, "Infinite Mario Bross AI using Genetic Algorithm," 2011 IEEE Conference on Sustainable Utilization and Development

in Engineering and Technology (STUDENT), Semenyih, Malaysia, 2011, pp. 85-89, doi: 10.1109/STUDENT.2011.6089330.

Name of Candidate: Jaran I. Holt

Date of Submission: April 21, 2025