Integrating a Human Feedback Loop in PCG for Level Design Using LLMs

Lucas van Koppen
Supervisor: M. A. Gómez Maureira (EEMCS-CS)
Critical observer: dr. M. Gerhold (EEMCS-CS)

July 23, 2024

Abstract

With the recent developments of Large Language Models (LLMs) and the technical capabilities of Procedural Content Generation (PCG), new opportunities arise when intersecting these 2 fields. Currently, there has been little to no research on the application of LLMs into PCG. This thesis explores some of these opportunities by researching how the integration of a Human Feedback Loop using LLMs improves PCG techniques in level design for games. To answer this question, a Super Mario Bros generator that uses constructive PCG was taken and the concept has been applied, allowing users to give feedback in the form of text and letting the prototype generate more personalized levels. User evaluations have been conducted to test the performance of the concept and find its strengths and weaknesses. In general, the concept works and shows promise as a tool for level design that can improve efficiency and usability, but using it as a game mechanic for existing games can prove to be challenging. As the capabilities of LLMs improve, the applications for the concept become more interesting within the field of game design. The results of this research show that the application of LLMs in PCG is promising and there are more opportunities for leveraging the interpretive capabilities of LLMs in the game industry.

Contents

Figure 1: Generated Minecraft (2011) World

1 Introduction

In the past 2 decades, the gaming industry has continued to captivate an increasingly diverse and expanding audience, while advancements in computing power and AI technologies push the boundaries of what's possible in digital environments. One of these new technologies is the development of Large Language Models (LLMs) like ChatGPT. This rapid evolution opens a wide array of opportunities for leveraging these technologies in novel and impactful ways.

This thesis explores an intersection of AI and game design by asking the following question: How can the integration of a Human Feedback Loop (HFL) using LLMs improve Procedural Content Generation (PCG) techniques in level design for games? PCG, usually utilized to automatically generate game elements such as levels, items, and narratives, contribute greatly to the replayability and variety of games. By integrating LLMs into the PCG framework, this research aims to make an application that can interpret user inputs expressed in natural language, allowing for the customization of game levels according to player preferences and desires. This concept will take the shape of an HFL, where a back-and-forth between the user and the computer creates an environment for the co-production of levels. This is an example of a Mixed-Initiative system, which is a form of PCG where the user and computer work together to create game content.

The vision of a game that generates content autonomously and tailors this content to individual preferences is a big step for the gaming industry. The advancements of PCG techniques can be seen in the success of major titles such as "No Man's Sky"(2016) and "Minecraft"(2011, Figure 1), which have captivated audiences with their vast, procedurally generated worlds. However, the integration of user-directed customization through advanced LLMs like ChatGPT represents an underexplored frontier with considerable potential.

Chapter 2 provides the current research on PCG, whereas chapter 3 depicts the applied methodology of this thesis. In chapter 4 the realisation of the research is presented. The research and its outcomes are evaluated in chapter 5, discussed and reflected upon in chapter 6 and concluded in chapter 7. Finally, in chapter 8 the recommendations deriving from the research are given.

2 Background Research

In this chapter, the proposed idea of an HFL is explained and the current body of knowledge on PCG and LLMs within game design is analysed.

2.1 The concept of a Human Feedback Loop

This research proposes the idea of an HFL to improve PCG techniques. The idea is to create a function for a game that utilises PCG, that allows the user to provide feedback on created content. Then the algorithm can learn and adapt using the provided feedback and potentially create more personalized content, improving the quality of procedurally generated content within the game (Figure 2). By repeating this process over many iterations, the algorithm will get closer to the preferences of the user. This implementation can be applied to many different cases and can take many different shapes. Some options and approaches are discussed here.

First of all, it can be applied to the creation of different types of game content, like the generation of worlds, narratives or any imaginable game object. This research focuses on the creation of levels for several reasons:

- Levels have a great impact on the experience of the user since levels generally have a big influence on the difficulty and gameplay. This makes testing if the application works well easier and if the application works, it can greatly improve the experience.
- Levels for simple games usually do not take too long to generate, if, for instance, you want to test it on the generation of worlds, it might take too long for the iterative process to have any meaningful effect.
- Usually, users play multiple levels after each other. This means its format is well suited for an iterative process. After playing the level, you can potentially give feedback to the algorithm, and let it generate another level which you can play again.

Secondly, it can be applied to different types of PCG techniques. As discussed in this chapter, several PCG techniques are worth researching for the integration of an HFL with the use of an LLM. Which PCG technique can be researched depends on the availability and quality of existing work. The most important aspect of the PCG technique is that there is some type of input that has a noticeable and somewhat predictable effect on the output and the possibility of an LLM being able to change these values.

Thirdly, the shape of the feedback loop itself can differ greatly. This is highly dependent on the game or PCG technique this concept is applied to. For a model using Reinforced Learning, some concepts were discussed, such as whether the algorithm produces only 1 level or multiple. Generating only 1 level will take less time, which will make the iterative process faster, but getting meaningful feedback for the algorithm is harder. Either the user has to specify what they like or dislike on that specific level, which the user might not even know themselves, or the program somehow needs to compare each iteration with the previous ones, either through the gameplay or directly asking the user. Generating multiple levels might take longer, but allows for the comparison between levels. This can make the feedback process easier for the user, as it is easier to tell the program which level you like or dislike more instead of telling it what you like or dislike about a level. This option does require the algorithm to learn what specific elements the user values, which can be challenging. This means it is a trade-off between how much the user has to do and how much the algorithm.

To better explain this, an example can be taken by applying this feedback loop to the generation of Super Mario levels. After the algorithm generates some levels, the user chooses which one it likes best, then the next iteration of generated levels should have some elements of the level the user liked. After selecting its preference a few more times, the algorithm now has a selection of levels that the user likes. Now it should somehow see what those levels have in common. Maybe it is the amount of blocks, or maybe it is the path the user has to take to complete the level. After many iterations, it should have a better and better understanding of what the user likes. If you were to directly ask the user what they like or dislike for each level, and the user for example says “This level has too many enemies”, then the algorithm is a lot simpler, which might make it more suited for testing the concept. The more simple the feedback the user has to give, the more the algorithm has to make up

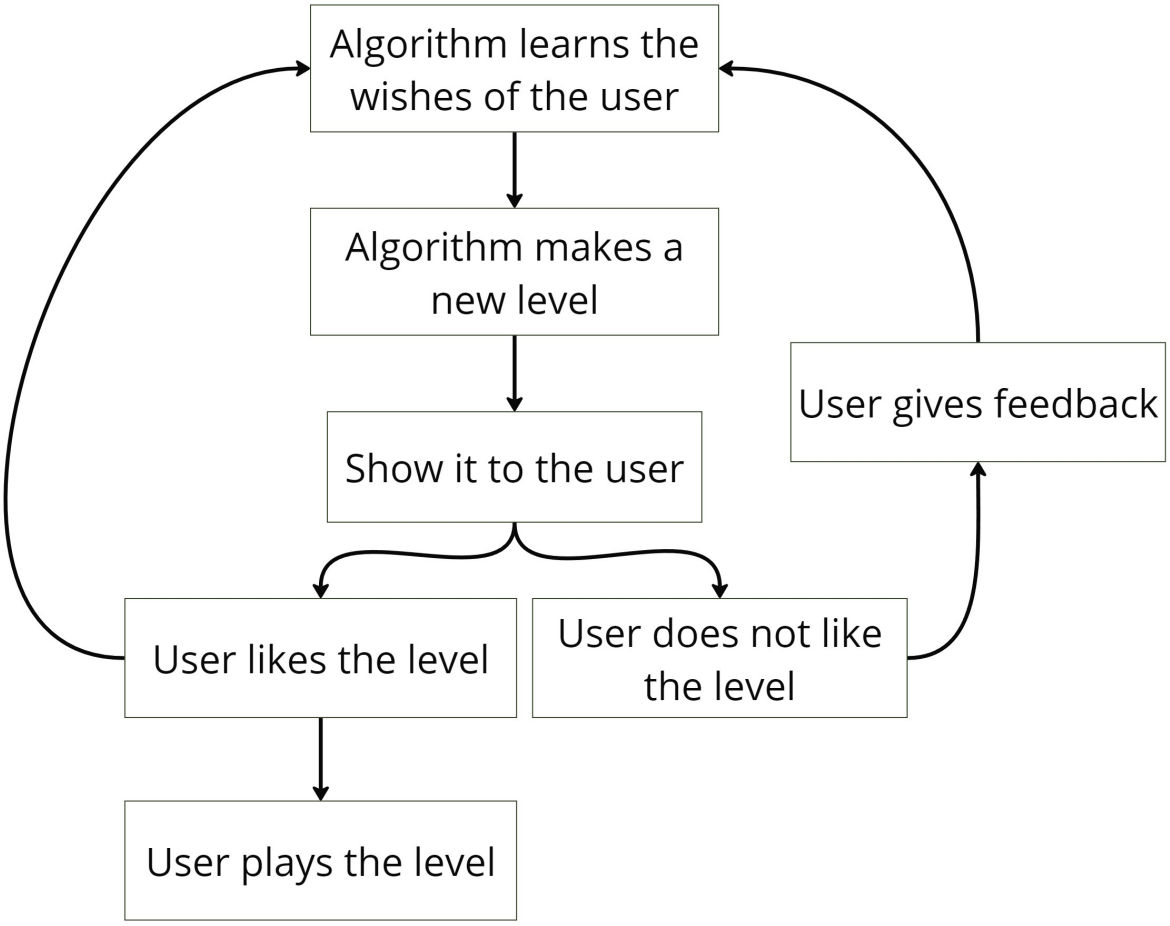

Figure 2: Diagram explaining the HFL

for it. Something that was touched upon a little already is the way the algorithm gets feedback. You can ask the user for its preference, or to describe what it likes or dislikes in words, but you can also let the user give a grade to a generated level, or let it answer some questions like "Does this level contain enough blocks for your liking". These are all ideas that can be researched.

Lastly, if prompts are received from the user to make personalized game content, this HFL can not only be used as a way for the algorithm to learn the preferences of the user, but it can also improve "reading" the prompts. So if for example, a user gives the algorithm the prompt that it likes many blocks and enemies in their level, then with the help of this concept, the algorithm can learn how many blocks and enemies this specific user, or users in general mean with its prompt.

2.2 Analysis of Mixed-Initiative Systems and the applicability of the HFL for PCG

Manually creating game content is a time-consuming activity and requires hard work from experts. Procedural Content Generation exists to help with the process of creating game content, exploiting the strong computational capabilities of modern technology, while also having the potential to enhance game experiences [1]. The capabilities of the recently developed Large Language Models (LLMs) give a very powerful tool to game designers: the ability to interpret and generate human text. The goal of this research is to explore the possibilities of enhancing PCG techniques with the help of LLMs and player feedback in the form of preferences.

Before exploring new concepts, it is important to analyse the current body of knowledge in this field. Since this concept can be applied to many of the existing PCG techniques, it is important to analyse how these methods work, what their benefits and challenges are and how well they are suited

for this concept. This overview of related work will not go into every PCG technique, as that would be too extensive. Instead, it categorizes them and focuses on the most important and relevant PCG techniques. This overview will only analyse PCG techniques in the context of game content generation, with a focus on level generation, since the concept will be applied to level generation. This overview will also discuss the challenges and benefits of Mixed-Initiative PCG (MIPCG) to better analyse the suitability of existing PCG techniques.

2.2.1 Mixed-initiative systems of PCG

Before discussing the various PCG techniques, first, some concepts of MIPCG are explained. To quote N. Shaker et al. [2]: "Mixed-Initiative PCG covers a broad range of generators, algorithms, and tools which share one common trait: they require human input in order to be of any use. While most generators require some initial setup, ... mixed-initiative PCG automates only part of the process, requiring significantly more human input during the generation process than other forms of PCG." Within the concept of MIPCG, there is a distinction between computer-aided design, where humans have the idea and the computer supports their creative process, and human-aided PCG, which is a system where the computer creates content and humans guide it to content they prefer. [2] This research is interested in the latter.

Since MIPCG is an umbrella term and can be applied to many of the PCG techniques, it is important to establish what the challenges are within MIPCG and thus what traits are valued for a PCG technique. Firstly, the input the algorithm gets from humans should have a noticeable effect on the output. The more impact the input has on the output, the better, as this means the PCG technique is adaptable. Secondly, the feedback process should not cause user fatigue. This can be achieved by making the decision process for the user easier by providing not too many options, asking few questions and making the decisions the user has to make as simple as possible [2]. This means it is optimal if the algorithm can function well with little input. Thirdly, the faster the algorithm, the better. If the generation process takes too long, players will lose interest in the process [3]. Lastly, it is desirable that if the user changes their mind about a preference, the algorithm can adapt. The proposed concept is creating an HFL using LLMs to interpret the preferences of users and translate them into input for a PCG algorithm. This concept has the potential to improve PCG techniques that need a lot of input and would cause user fatigue if used traditionally. The goal of this overview is to find PCG techniques that would benefit from the implementation of an LLM and an HFL.

2.2.2 analysing PCG techniques for suitability of the concept

Now that the desirable traits are discussed, the first PCG category can be analysed, namely search-based PCG. According to J. Togelius et al. [4], Search-Based PCG is a special case of the generate and test approach to PCG, with two qualifications. Firstly, instead of accepting or rejecting candidate content, the test function grades all generated content, which is called a fitness evaluation. Secondly, generating new candidate content is dependent on the assigned fitness values of previously evaluated content. The most common search-based approaches are variations of evolutionary algorithms. These algorithms contain a population of candidate content where each generation, candidates are evaluated, the worst candidates are discarded and replaced with copies of good candidates that have been randomly modified.

An essential part of this process is the evaluation phase, where human feedback can be used to improve the algorithm. Togelius and Yannakakis [5] mention that it is essential that the candidate content is evaluated correctly and that this evaluation, especially when it comes to subjective metrics like how "fun" or "immersive" generated content is, poses to be challenging. Here is where the capabilities of human evaluation can be leveraged. By replacing the fitness evaluation with human evaluation, i.e., the human decides what candidate content is "good", the challenge of subjective evaluation is resolved. This form of MIPCG is called interactive evolution and is the most common form of human-aided PCG. There have been many studies done on level generation through interactive evolution, some notable researches are [6–10]. D. Gravina et al. [11] mentions that their developed quality diversity algorithm is an especially effective tool for MIPCG, as it provides high-quality and diverse suggestions. As interactive evolution is a system that conforms to a lot of the desired traits in MIPCG, it is no surprise this is the most common technique within human-aided design. Since there

is a lot of research done in this field and this approach would not benefit a lot from the usage of LLMs, this technique will not be researched further for the implementation of the proposed concept.

The second PCG category that will be analysed is machine learning-based PCG. To quote A. Summerville et al. [12]: "We define PCG via machine learning (PCGML) as the generation of game content by models that have been trained on existing game content. The difference between search-based and solver-based PCG is that, while the latter approaches might use machine-learned models (e.g., trained neural networks) for content evaluation, the content generation happens through search in content space; in PCGML, the content is generated directly from the model." As there are a lot of PCG techniques that fall into this category, two concepts within PCGML that show the most promise will be analysed here.

The first concept to discuss is reinforced learning (RL), which is a concept within deep learning. According to Otterlo and Wiering [13], RL is a type of machine learning technique where an agent learns how to behave in an environment by performing actions and seeing the results of these actions. The learning process involves the agent understanding what actions lead to the best long-term benefits through a system of rewards and penalties. The goal of the agent is to perform actions that maximize the reward signal in the long run. A. Khalifa et al. [14] mentions that RL in level design is much faster in generating levels after training than search-based methods. The training process however takes considerably longer. They also mention that due to the incremental nature of the trained policies, they are suitable for MIPCG. Since RL works with a reward model, humans can potentially improve this system by giving feedback on the policies for the reward system, or by directly providing rewards to the system. LLMs can help with this process, as they can translate the preferences of the user to these policies or rewards. Apart from the fact that once the RL is trained, it will be hard to change what it has learned, meaning the last trait of adaptability after a preference change will not be satisfied, it is very well suited for the proposed concept.

The second concept to discuss is supervised learning. According to Cunningham, Cord and Jane Delany [15], supervised learning entails learning the connection between a set of input variables and an output variable and applying this mapping to predict the outputs of new data. This is not so much a PCG technique, as it is a learning method for various algorithms within machine learning. This concept is highly dependent on labelled data to learn the mapping of inputs and outputs. Labeled data is hard to come by, as it requires a lot of time and energy to label a lot of data. LLMs have the capabilities to help in this process, by translating human feedback into labels. This way the algorithm can learn to predict what type of levels the user prefers. Although this method might also struggle to adapt when the user changes preferences, it is well suited for the proposed concept.

The last PCG category to analyse is constructive PCG. According to Togelius and Yannakakis [5], this is a very broad term containing every variation of an algorithm that generates the content in one go. In this process, the algorithm needs to make sure the created content is correct as it is being constructed. This means algorithms within this category must adhere to rules and limitations set up by the designer, as a means to always generate correct content. LLMs can be used to translate the preferences of the user into such rules and amplify the likeliness of certain content being generated during the process. A benefit of this technique is that the proposed concept is relatively easy to apply. The downside however is that the algorithm will not have a deep understanding of underlying patterns in the preferences of the user, something techniques within PCGML are capable of. This technique might be a good starting point for the research to check if the concept works.

2.2.3 Essential findings and related work on additional PCG techniques

As only a few PCG techniques are discussed, there are still some approaches that are worth shortly mentioning as they can also benefit from the proposed concept. For systems that use general advisory networks, like [16], LLMs can be used to improve the discriminator. For systems that use solver-based techniques, like [17], LLMs can be used to formulate constraints for the algorithm. There are also many possibilities within hybrid methods of PCG, which is a combination of different PCG techniques combined in one algorithm. Through hybrid systems, the challenges of some PCG techniques can be reduced by leveraging the benefits of other techniques. There are also some interesting papers on the computer-aided design part of MIPCG [18–20] and experience-driven PCG [21, 22] that have points of intersection with this research.

To conclude, the challenges of MIPCG have been discussed, some relevant PCG techniques have been analysed, looking at how they work, what some challenges are and how they could benefit from

introducing LLMs as a means to translate user preferences to the algorithm. Search-based systems appear to not benefit greatly from the proposed concept, while reinforced learning, supervised learning and constructive PCG all show promise to improve from the proposed concept. As the development of LLMs is very recent, there still is a lot of room for further research on LLMs within the field of PCG.

2.3 Analysis of Large Language Models in PCG

LLMs have shown to be a very successful technology that is widely used by a great variety of users. They have been used primarily for generating human text. It is a very powerful tool for writing reports, answering questions, interpreting text for the generation of images, restructuring and summarizing various sizes of text, and it has many more uses [23]. This tool however still has a lot of undiscovered uses. Its capabilities also show great promise within the game industry. A very obvious user case for this tool within the game environment is for generating narratives and stories. Since the input and output are text, you can quite directly implement it as a feature to tell a story or to handle conversations. There have been studies done on this application, some examples that have researched this application of LLMs [24] [25] [26] have shown success. You can however also use it to make less obvious content, such as levels, worlds, rules, items, aesthetics or any other game content. The use of LLMs to create such content does come with restrictions and challenges. Since this technique is a form of PCGML (Procedural Content Generation via Machine Learning), you need large datasets to "train" the algorithm. Moreover, you also need to process the content into data that you can give the algorithm, and translate it back to the content afterwards. This can prove very challenging for some types of content, for example, if you want to generate a game world, since this content generally does not contain big datasets, and processing a whole world into data the algorithm can use is very hard and time-consuming. It is however possible for LLMs to generate levels for simple games. Levels are usually a 2 or 3-dimensional space that a player-controlled character must somehow traverse, often while accomplishing other goals [4]. To research if this works well, it is best to use 2-dimensional levels, since the data is simpler and the datasets are usually bigger. A bit of research has been done on this application for simple 2-dimensional games, like for the puzzle game Sokoban (figure 3) [27] and Super Mario (figure 4) [28], and since their research proves this application to be successful, it is interesting to research whether LLMs can be used as a PCG technique to generate different game content as well.

The usage of LLMs within PCG has more benefits, not only can you directly use it as a PCG technique, but you can also use it to interpret user input in the form of text. Successfully implementing this feature would mean users can get access to unlimited personalized game content. The study on Mario also researched this and again proved successful. These are very exciting results for the game industry, as the promise of unlimited personalized game content is a dream of many gamers. There are however still a lot of things to do to realise that dream. One interesting research might be to see if you can apply this also to 3-dimensional levels, and if you can use the interpretation capabilities of LLMs on different PCG techniques. For example, Trackmania Turbo(2016, Figure 5) has a random track generator, which builds tracks in a 3-dimensional plane using constructive generation techniques. This could be an interesting next step for the use of LLMs within the field of PCG.

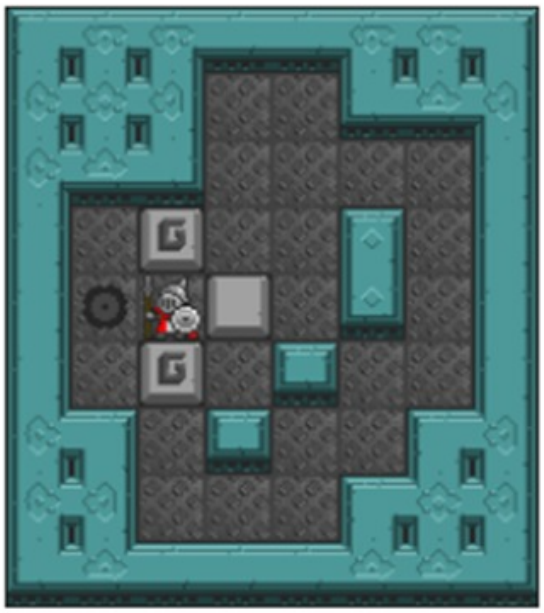

Figure 3: A Sokoban level generated by an LLM [27]

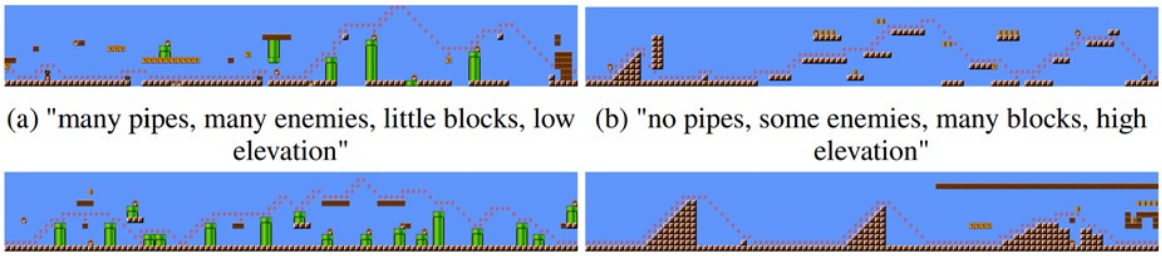

Figure 4: Explanation of MarioGPT [28]

Figure 5: Generated track for the game Trackmania Turbo

3 Method for the design and testing of the HFL

In this chapter, the approach of the research is explained, the stakeholders are defined and the reason for evaluation is clarified.

3.1 The approach to research

The project is done by a research team, with a student and 2 supervisors. During the development of this project, this team collaborated and brainstormed collectively to come up with ideas and solutions for problems that arose. The approach for this research is first to come up with a general research topic and to look for problems or opportunities within this general direction. The goal here is to find a relevant topic that can contribute to the research within the field and slowly specify the direction as to get to a more concrete plan. After finding a relevant and specific topic, the next step is to read into the current body of knowledge related to the specified direction. This is necessary to be informed about the current state and advancements around the topic, and to define how this research can contribute to and expand on current research. After the current body of knowledge is analysed, a plan can be devised to check the hypothesis or answer the research question. This plan can change throughout the research, as some things can go wrong or turn out differently than expected. In that case, several approaches can be taken to change the plan or solve the problems. This is always done in discussion with the research team. After devising and carrying out the plan, the process is evaluated and reflected upon, and a conclusion for the hypothesis or research question can be formed.

The results of each step are covered here. The general direction for this thesis was programming and games. The direction was further specified to PCG, which is an intersection of these 2 fields. Within this direction, an idea sparked forth from the research team: Utilise human input to improve PCG through a feedback loop. During the analysis of the current body of knowledge, specific PCG techniques were analysed for their compatibility with this idea. The background research concludes that supervised learning, reinforced learning, constructive PCG, and potentially other PCG techniques are well suited for the idea, but the concept of applying LLMs in level design is under-researched and lacks scientific investigation. During the analysis, a particular study, MarioGPT, came forth as an interesting study to build upon. This study was closely aligned with the specified topic and was well-suited for applying the concept. The plan was to try to apply the concept of an HFL and improve the algorithm, letting it learn the user preferences and generating more personalized levels. After finding out that this study was in fact not suited for the concept, a step back was taken and a new plan was devised. This time, a different PCG technique, constructive PCG, was used. Since applying the concept was easier for this PCG technique, a LLM was used to translate human text to variables the level generator can use, allowing users to use text as input for the HFL. After building this prototype, user evaluations were conducted, the process was reflected upon and a conclusion was formed.

3.2 Stakeholder Analysis for the impact on the industry

To further emphasize the rationale for this research and to have a better vision of the impacts within the industry, it is important to look at the stakeholders. Given the project's focus on technologies within the gaming industry, some stakeholders have been identified, including:

- Gamers, who interact directly with the game, experiencing and influencing the research developments firsthand.
- Game Developers, who might utilize this technology either as a feature to enhance gameplay or as a tool for streamlining level design.
- Game Publishers, who could leverage this advanced PCG technology as a marketing tool to attract a broader audience.
- Researchers, who could build on this foundational work to push the boundaries of personalized content generation within the gaming industry.

3.3 Proof of concept

For this research, it is important to test how well the concept of improving PCG with an HFL using an LLM works. Not only will it improve this research and its application, but it is also important to look at what works and what doesn't to help future research. To assess how well the concept works, firstly you can look at the application and look at the facts while answering these questions. Does the program work? Are the levels playable? Are there any problems with the format of the response? Do the changes in values accurately represent the feedback and make sense? By asking these questions you can conclude whether or not the concept works from a practical standpoint. If you however want to know how satisfied the users are when using the HFL, or if the effects are desirable, then you need to conduct user evaluations. Conducting user evaluations has 3 purposes:

- It gives more insight into the effectiveness of the concept and its effect on the user. By asking questions about satisfactions, feelings and thoughts that arise while interacting with the product, you can see what effect the concept has on users. This can prove whether the concept is beneficial and it can show what might be challenges when doing further research.
- It gives insight into what type of input different users will give to the program and how ChatGPT will respond to it. This is also very meaningful data since it shows the flaws of the concept, which allows you to tweak and improve the website based on the results. It also proves whether or not the prototype works in different situations. with different users.
- Users might have new perspectives and suggestions to improve either the functionality of the prototype or the concept. By explaining the project and design choices afterwards, users might give valuable insights and ideas about the concept that have not been considered before.

4 Realisation

In this section, 2 attempts at applying and realising the concept are discussed. The first plan was an attempt to continue with the study of MarioGPT and the reasons why the plan did not succeed are mentioned. The second plan was an application of the concept on a Super Mario Bros generator for the browser [29]and the steps taken to realise the concept are discussed.

4.1 Plan 1: Applying the concept to Reinforced Learning

The first step of the plan is to find existing work that can be built upon, as it is too much work and not the point of the research to make the generator itself. When exploring studies that combine LLMs and PCG, the study of MarioGPT [28] was found. They have made a program that creates Super Mario levels using LLMs both as a PCG technique and as a way to interpret prompts. This makes it an interesting study to build upon. It is well suited for applying this concept of a Human Feedback Loop for multiple reasons. Firstly, it is open source, so you have easy access to everything they made. Secondly, the game it makes levels for is fairly simple, which makes it well-suited for research of concepts. Thirdly, since the study already integrated interpreting user prompts, you can research if the HFL works for improving the "reading" of prompts. Moreover, the levels can be tested quite easily and the used PCG technique can be altered by the iterative process. The paper mentions in its conclusion that the author also is interested in implementing an HFL.

While analysing the code to see if some of these concepts can be tested, there were some challenges that arose:

- The code does not have a lot of comments and it is hard to get a full grasp on what piece of code does what exactly. There are a lot of **unexplained and complex processes within the code,** that are hard to understand when you are not involved in the development of the program.
- The **generation process of the levels takes a long time**. This means if you want to make an HFL that shows multiple generated levels, the user needs to wait a long time in between every iteration, which is sub-optimal for research or for a final product. The generation time can be decreased by decreasing the size of the level, but that comes at the cost of quality of research.

- **There were no variables that had an impact on specific elements of the level.** In an attempt to find out if applying the HFL to improve the "reading" of prompts is possible, the piece of code that handles the prompts was analysed and some numbers were changed to see how it would affect the level generation. After running many iterations, it became apparent that changing the numbers did very little. In the paper, they mention that the algorithm performs most accurately on the generation of blocks, so the analysis has been done on the amount of blocks generated. In the Appendix, you can find figures showing the input and output of the program. As you can see, 2 lines of values can be changed per aspect. It was expected that these numbers have something to do with the amount of generated content of that type, representing "few", "some" and "many" of that type respectively. But when changing these values, there is no noticeable effect on the generated level. In most cases, no matter the input, it can generate levels with many blocks or very few blocks. For each change of the set of inputs, 2 outputs are shown that show a great difference in the amount of blocks. This would suggest that changing the numbers in the code does not have an effect, or that the effect is not always noticeable.

The last point is a real problem. If the program cannot reflect changes based on the input accurately, then it will be very hard to continue working on this project. It is common for AI models to have this challenge, **where it is similar to a "black box"**. You put some values into the model, some unexplainable or hard-to-grasp processes happen, and you get an output that is (hopefully) useful or reflects what you had in mind. However, if the output is not useful, or does not accurately reflect changes based on the input, then it will be very challenging to pinpoint the cause of the problem within this "black box". As this project came to a halt because of this, a new approach was necessary to research the implementation of an HFL.

4.2 Plan 2: Applying the concept to constructive PCG

Atructive s the first plan ofcons MarioGPT had too many challenges to overcome, the plan had to be changed. As mentioned in the background research, constructive PCG techniques can also benefit from the integration of an LLM and a feedback loop. The new approach is to research the possibilities of the feedback loop within constructive PCG since this PCG technique is easier for implementing the concept, and although the program will have a less in-depth understanding of the preferences of the user, it is sufficient for testing the concept. This approach will also only take human text as input and use an LLM to translate it to terms the PCG algorithm can understand. Both of these changes make this plan less challenging than the last one. The LLM that will be used for this research is ChatGPT by OpenAI. The reason to choose ChatGPT is the same reason why the program is so popular: their API is straightforward and works well, their documentation is clear and extensive and the quality of the output is (one of) the best currently available.

The first step of the plan again is to find existing work that can be built upon. The goal is still to apply it to level design, and since Super Mario is a popular and effective research game, it was decided to search for a constructive Super Mario generator. Some existing works have been found and analysed, like [30], [31] and [29]. After a quick analysis, it was found that the work of [29] was well suited for this project, as the code was very clear and understandable, there were enough variables for the LLM to change and the project was made in Javascript, so it can be played in a browser. This means it is more accessible to test, and communication with the ChatGPT API will be possible.

Now that a level generator has been selected, the next step is to take a close look at the code and understand where and how the levels are generated. The levels are generated by a lot of random functions, that depict things like the size of a gap, the odds of a question mark block appearing or the spawn points of enemies. All these odds of something happening clearly affect how the level looks, so these are the perfect "knobs" for the LLM to change according to the preferences of the user. After understanding how the levels are generated, all these odds were changed to variables, they were analysed to see what exactly it changes and a short description was added to each variable explaining what it does and how it is used. The restrictions were also analysed and noted down, for example, the gap distance should never be able to be greater than 4, as this could result in an unplayable level. An explanation of the context was added together with the restrictions and explained variables. Putting this combined text into ChatGPT and giving it feedback on what should be changed in the level already gave good results. If you copied the output of ChatGPT directly into the code, you would get a level that more accurately represents the preferences based on the feedback. This already proved

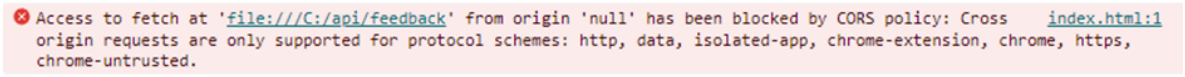

Figure 6: CORS error message

that the concept has promise, but user tests need to be done to get a more accurate understanding of what works, what does not, how the user interacts with the program and how the program will react to the different inputs it gets. To do these user evaluations, the feedback loop must be as direct as possible. This means that ideally, the user can fill in the feedback in the same application they play the level in and that it directly updates the response of ChatGPT into the level generator.

The next step is to create an environment where the feedback loop can take place and create connectivity with the ChatGPT API. The code that is provided on GitHub not only generates a random level, it also generates a random Super Mario world with different levels. Since the feedback loop consists of the user playing a level, giving feedback and playing a level again, the implementation of a world is not well suited. This is why this feature is removed and the user directly plays a level when they press play. It might be interesting to add this feature after the program has "learned" the preferences of the user, allowing them to work towards a bigger goal. The connectivity with the ChatGPT API posed its challenges. If you run the program directly from your local files in the browser and try to access the OpenAI endpoint, it runs into problems with CORS (Figure 6). This is a measure to protect websites from accessing your local files. To fix this problem, you need to run a server and access the endpoint through it.

The GitHub page function was explored to fix this issue. You can post code on GitHub, and let it run a site on a server of them. But this posed another problem. To access the ChatGPT API, you need a code that tells you who is making the request. Since their services are not free, this code must stay secret. By using the GitHub page function, the code will be made public and will be immediately disabled by OpenAI. So you need to host a server locally somehow, which can be done using Node.js, but this is a tedious task, especially for someone with little to no experience in web development. There is a program called Xampp that can create a server for you. This managed to resolve the CORS issue, without sharing the secret code.

OpenAI provides 3 options for API requests, you can use Curl, Python or Node.js. Since this application is made in a browser, the Node.js approach was needed. To make a request to ChatGPT, you need an OpenAI module. This can be downloaded via Node.js, but when running the program in the browser, you run into some problems again. You need to specify that the JavaScript file that uses the module is imported as a module in the HTML script (by typing type=module). Also, the package cannot be found when opened in a browser, as it runs into CORS problems again. To resolve this issue, the package can be directly downloaded in the browser by using Skypack [32].

The next step is adding an input box and a submit button to receive and send the feedback. When you submit the feedback, the text is removed from the input box and moved to a log/chatbox that the user can see. Before providing the feedback, you of course need to explain to ChatGPT what it needs to do, and what the format of its response should look like. This explanation is put in a txt file, which is read by the script and sent to ChatGPT when the website is loaded. It gives a short explanation of the context, a list of variables with explanations in JSON format, and afterwards a few restrictions. It asks ChatGPT to only respond with the list of variables in JSON format, which then is directly put in the code and used by the level generator. It takes a few seconds for ChatGPT to respond, so the variables are updated the moment it receives a response. ChatGPT does not remember the conversation with you, so you need to store that conversation yourself and send the whole conversation history every time you make a request. This is also useful when performing user tests, as keeping track of the back-and-forth of the feedback loop can provide new insights.

Lastly, the response of ChatGPT might be in JSON format, but it is not yet seen as JSON, so you need to parse it to JSON for the level generator to correctly read it. The program runs successfully and creates playable levels that reflect the feedback provided. The next step is to change the random functions into seed functions, as a way to increase reproducibility. This seed can be implemented using a module from Skypack [33]. This module gives access to a function that can globally seed all the math.random functions. By setting this seed right before the level generated, it is possible to recreate the level that was played, given you know the seed number and configuration settings provided by

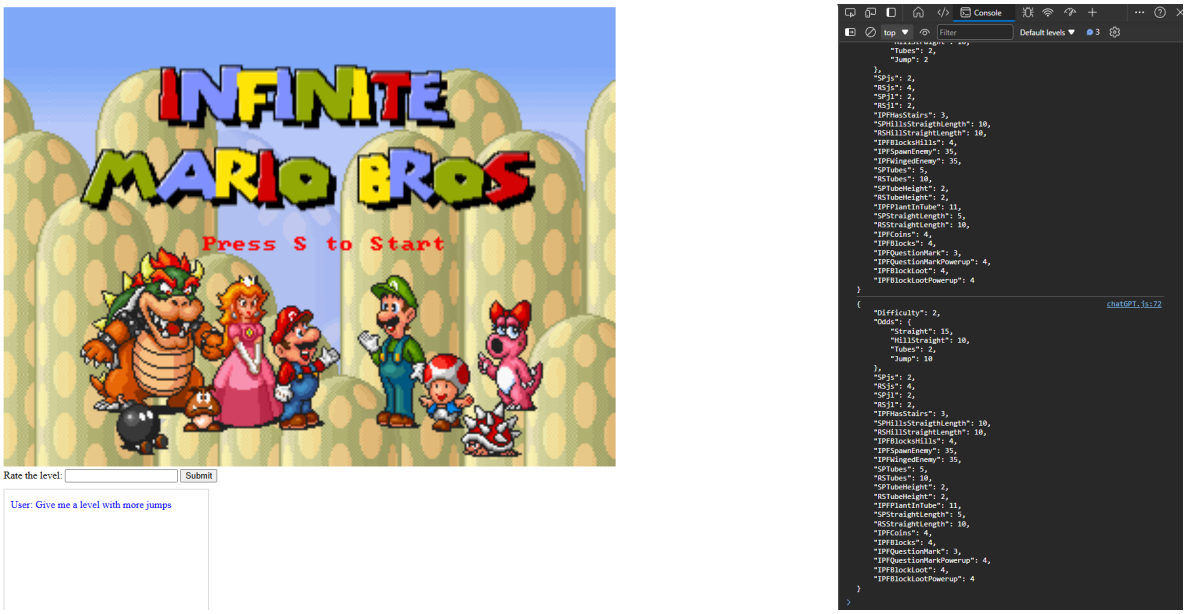

Figure 7: The product of plan 2 with the responses in the console.

ChatGPT. The prototype is now functional in the browser (Figure 7) and the first and final version of the explanation prompt are provided in Figure 8 and 9.

In the process of making this application, it happened a lot that the code was altered and saved, and the browser was reset, but the code actually did not change in the browser since it remembers the old code in the cache. That is why while working in web development, it is important to disable the cache in the browser, so the browser always has the latest "version" of your code.

```
There is a Super Mario This is a random level generator which has variables for
generating the level. I let a player play a random level, and let the user give you
feedback.
Then, I want to you to change some of these values to better suit the preferences of
the user. When you respond, you should only respond with the list of variables without
the comments, as it is in json format.
dont provide any additional context, as i will directly place your response in the
code. Here is the explanation of the variables:
        Whenever Range Size (RS) or Starting Point (SP) is mentioned, it means this
number is used in a function of (Math.random() * RS) | 0 + SP
        to determine the size of something.
        Whenever the Inverse Probability Factor (IPF) is mentioned, it means this
number is used as the chance of something happening in a boolean
        statement, for example: Math.random() * IPF) | 0 === 0. A lower number means
higher odds of that thing happening, with 0 being the lowest possible number.
        IPF can be floats, RS and SP need to be integers.
        This class generates a level by combining different types of zones. The odds
of generating a zone of a certain type can be changed here.
        The higher this number, the higher the odds are of that that zone will be
chosen next.
        list of variables that can be changed:
{
    "Difficulty": 2,//minimum 1, increases by 1 for each level completed impacts how
many enemies spawn, what zones are built
    "Odds": {
        "Straight": 20,
        "HillStraight": 10,
        "Tubes": 2,
        "Jump": 2
    },
    "SPjs": 2, //SP of free space before a jump
    "RSjs": 4, //RS of free space before a jump, also influences the height of stairs
if they are generated
    "SPjl": 2, //SP of the length of the jump
    "RSjl": 2, //RS of the lenght of the jump
    "IPFHasStairs": 3, //IPF of generating stairs for a jump
    "SPHillsStraigthLength": 10, //SP of the lenght of a HillStraight zone
    "RSHillStraightLength": 10, //RS of the length of a HillStraight zone
    "IPFBlocksHills": 4, //IPF of spawning blocks in a HillStraight zone
    "IPFSpawnEnemy": 35, //IPF of spawning an enemy
    "IPFWingedEnemy": 35, //IPF of an enemy having wings when spawned
    "SPTubes": 5, //SP of the length of a Tubes zone
    "RSTubes": 10, //RS of the length of a Tubes zone
    "SPTubeHeight": 2, //SP of the height of the tube
    "RSTubeHeight": 2, //RS of the height of the tube
    "IPFPlantInTube": 11, //IPF of a plant enemy spawning in a tube
    "SPStraightLength": 5, //SP of the lenght of a Straight zone
    "RSStraightLength": 10, //RS of the lenght of a Straight zone
    "IPFCoins": 4, //IPF of coins spawning
    "IPFBlocks": 4, //IPF of coins spawning
    "IPFQuestionMark": 3, //IPF of a question mark block spawning instead of a normal
    "IPFQuestionMarkPowerup": 4, //IPF of a question mark containing a mushroom or
flower
    "IPFBlockLoot": 4, //IPF of a normal block containing loot (coin or shroom/flower)
    "IPFBlockLootPowerup": 4 //IPF of a block that contains loot, to contain a shroom
or flower
}   //There are some restriction to make the level playable, these are that RSjl +
SPjl and SPTubeHeight + RSTubeHeight should never be greater than 4. Start of by
giving this list without comments but with the same values.
```

Figure 8: The first version of the explanation that is sent to ChatGPT

```
This is a Super Mario random level generator which has variables for generating the level. I let a
player play a random level, and let the user give you feedback.
Then, I want to you to change some of these values to better suit the preferences of the user. When
you respond, you should only respond with the list of variables without the comments, as it is in json
format.
dont provide any additional context, as i will directly place your response in the code. Here is the
explanation of the variables:
        Whenever Range Size (RS) or Starting Point (SP) is mentioned, it means this number is used in
a function of (Math.random() * RS) | 0 + SP
        to determine the size of something.
        Whenever the Inverse Probability Factor (IPF) is mentioned, it means this number is used as
the chance of something happening. A lower number means higher odds of that thing happening, with 0
being the lowest possible number.
        IPF can be floats, RS and SP need to be integers.
        This class generates a level by combining different types of zones. The odds of generating a
zone of a certain type can be changed here.
        The higher this number, the higher the odds are of that that zone will be chosen next.
        list of variables that can be changed:
{
    "Difficulty": 2,//minimum 1, If the user wants a harder level, then you dont need to change a lot
for the other values, just increase this.
    "Odds": {
        "Straight": 20,
        "HillStraight": 10,
        "Tubes": 2,
        "Jump": 15 //all these zones are equally hard, just make sure tubes is not too high
    },
    "SPjs": 1, //SP of free space before a jump
    "RSjs": 4, //RS of free space before a jump, also increases the height of stairs if they are
generated
    "SPjl": 2, //SP of the length of the jump, if the user wants bigger jumps, increase SPjl by 1 and
decrease RSjl by 1
    "RSjl": 2, //RS of the lenght of the jump, if the user wants smaller jumps, decrease either SPjl
or RSjl
    "IPFHasStairs": 3, //IPF of generating stairs for a jump
    "SPHillsStraigthLength": 10, //SP of the lenght of a HillStraight zone
    "RSHillStraightLength": 10, //RS of the length of a HillStraight zone
    "IPFSpawnEnemy": 35, //IPF of spawning an enemy, if a user wants no enemies, make this number
really really big.
    "IPFWingedEnemy": 35, //IPF of an enemy having wings when spawned
    "SPTubes": 5, //SP of the length of a Tubes zone
    "RSTubes": 10, //RS of the length of a Tubes zone
    "SPTubeHeight": 2, //SP of the height of the tube, if the user wants higher tubes, increase
SPTubeHeight by 1 and decrease RSTubeHeight by 1.
    "RSTubeHeight": 2, //RS of the height of the tube, if the user want lower tubes, decrease either
SPTubeHeight or RSTubeheight.
    "IPFPlantInTube": 11, //IPF of a plant enemy spawning in a tube
    "SPStraightLength": 5, //SP of the lenght of a Straight zone
    "RSStraightLength": 10, //RS of the lenght of a Straight zone
    //The following values will make the level easier if you decrease them:
    "IPFCoins": 4, //IPF of coins spawning
    "IPFBlocks": 4, //IPF of coins spawning
    "IPFBlocksHills": 4, //IPF of spawning blocks in a HillStraight zone
    "IPFQuestionMark": 3, //IPF of a question mark block spawning instead of a normal
    "IPFQuestionMarkPowerup": 4, //IPF of a question mark containing a mushroom or flower
    "IPFBlockLoot": 4, //IPF of a normal block containing loot (coin or shroom/flower)
    "IPFBlockLootPowerup": 4 //IPF of a block that contains loot, to contain a shroom or flower
}   //There are some restriction to make the level playable, these are that RSjl + SPjl and
SPTubeHeight + RSTubeHeight should never be greater than 4. Give this list without comments and
implement the following feedback:
```

Figure 9: The final version of the explanation that is sent to ChatGPT

5 Evaluation

In this section, the approach for user evaluations and the results of the evaluations, which are divided by feedback on the functionality and the concept, are discussed.

5.1 Approach of User Evaluations

To evaluate whether or not the concept and the prototype work, user evaluations have been conducted. This has been done in the form of semi-structured interviews. For this, 12 students, ages ranging between 18 and 25, have been asked to participate. Most of these participants have experience with games, some have very little experience. The structure of the interview was as follows: First, people were briefed about the research. This includes an explanation of what the prototype does, what the controls and inputs are, why the interviews are conducted and what is expected of them. Then they are asked to interact with the prototype. During this interaction, any noticeable interaction and reaction of the user and the prototype was written down. Afterwards, some questions were asked. If necessary, any follow-up questions were asked to clarify or give more insight. These are the questions that were asked:

- What did you like about this way of designing the level?
- What did you dislike about this way of designing the level?
- What would you improve? What would you wish you could do?
- How do you feel like the levels matched your preferences?
- Would you be more likely to play a game that has a functionality like this built-in?

The last question was used to invite the participants to think about applying the concept to other games, and what type of games they would be most interested in. During these interviews, one additional question was asked quite frequently, which is: Would this concept be more suited for level designers or users? After the questions, the participants were debriefed. To better keep track of the interactions and input from users, the input prompts and responses from ChatGPT were copied after the interviews. This has been done for the last 8 interviews, as the benefit of this was realised only after the fourth interview. With the response of ChatGPT and the seed number, it is possible to recreate the levels that the users played for the last 8 interviews. The user evaluation covers both the prototype and the concept, so for the sake of clarity, the results are divided for the functionality of the prototype and the concept.

5.2 Evaluation of Functionality

This section covers how well the prototype functions and how it deals with feedback it cannot implement. Firstly, when asked "How do you feel like the levels matched your preferences?", most users mentioned that the prototype responded well and clearly provided a level that was more in line with their feedback. This means the prototype is mostly a success as far as the intended design goes. There were however cases where the prototype did not work as intended. These "failure" cases can be categorized into 2 scenarios:

- The prototype changes nothing
- The prototype does something wrong

The first scenario mostly happens when the program is faced with feedback about an element of the game that it cannot change, or it gets a prompt which has nothing to do with the application. For example, during the user tests some participants asked to change Mario to Luigi or to change the length of the level. Since these are elements that are not in the list of provided variables, ChatGPT responds with the same values for the variables as the last iteration. This usually does not pose a problem for the functionality of the prototype, however, the users were disappointed when they noticed their feedback was not implemented. When it was given feedback about changing the colour of the turtle to red, it added a variable called "TurtleRed" and set it to true. This did not affect the level, as

this variable is not used in the code. Scenarios like this can however potentially cause problems within the code for other projects and should be taken into consideration.

The second scenario is mostly caused by a wrong or incomplete understanding of the level generator and how the values are used. For example, when given the prompt "no enemies" as feedback, it changed the "SpawnEnemy" variable to 0. However, the lower the variable, the higher the odds are it spawns an enemy on a tile, so unlike what the user had envisioned, the level was full of enemies. For another user, without changing the prototype, it did change the number to 10000 when asked for "no enemies", so ChatGPT still is somewhat unpredictable. Another example is when asked for "more powerups", it only increased the odds of giving a power-up within a block, but as the odds for spawning a block were low, nothing noticeable changed. ChatGPT missed the understanding that for more power-ups to spawn, also more blocks needed to be generated. A last example is when asked to increase the difficulty of the level, it increased the odds of generating jumps and pipes, however on these "pieces" of the level, enemies cannot spawn, so the levels turned out easier. After the user mentioned it was easier and it should make it harder, it still increased the odds of spawning jumps and pipes, while also increasing the odds of spawning enemies. So since ChatGPT does not understand exactly how the levels are generated and it is not capable yet of understanding the whole code, it makes some mistakes.

Lastly, there were some general suggestions/improvements for the prototype. Since it takes about 6 seconds for ChatGPT to respond to the feedback and give a list of variables, it would be nice to show a loading screen or to notify the user in some way in the interface that feedback is implemented. Someone mentioned it would like some type of leaderboard to measure how well you did and a place to share and play levels from other players. Another functionality a user mentioned is that they want to change specific parts of the level. For example, first, give me a power-up and then give me a lot of enemies. Lastly, someone mentioned sometimes the configuration turned out worse after the feedback, so it wanted an "undo" button to go back to the previous configuration.

5.3 Evaluation of Concept

This section covers how well the concept works and describes its strengths and weaknesses. In general, the participants were excited when they noticed their feedback got implemented. This can be either because they did not expect it to work, or because they genuinely got excited about the concept. Either way, it shows the concept has a positive initial reaction. Another thing that can be noticed from the user testing was how little the feedback loop was actually used by most users. They focused more on playing the game than typing feedback. One user mentioned how, if you want users to use the HFL more, it should be more "gamified", as the current application is quite dry. Also, users almost always wanted to complete the first level before using the feedback mechanic.

Some users mention the lack of purpose. They miss a goal, some type of progress or a bigger thing to move towards. Some mentioned they want a leaderboard to see how well they did in comparison to other players, and this could work if you see the application as a tool to design levels. When asked if they think this concept is more suited for level designers or directly for users, almost everyone thinks it can be a useful tool for both.

The last common thing most users wanted was the ability to change more in the game. Some examples of what users tried and would have loved to see implemented in the game are things like getting the ability to fly or giving Mario a glider. Another popular aspect users wanted to change was the aesthetics of the game. People asked to change the colours of Mario or the enemies or change the background of the level.

6 Discussion

In this section, improvements for the prototype, the different applications for the concept and a reflection on the project are discussed.

6.1 Prototype improvements

During the user evaluations, there were a lot of problems that arose. Most of these problems can be fixed and would greatly improve the prototype. Implementing these functionalities will be beneficial for future research and elevate the possibilities of the concept. The improvements will be categorized in the same way as they were categorized in the evaluation phase.

In the case of the prototype not doing anything, several solutions can improve the situation. Firstly, ChatGPT should be able to defend its choice for the new variables. So if for example, the user asks to change Mario to Luigi or asks the increase the length of the level, ChatGPT can respond with "Sorry, but I cannot change this aspect of the game", which increases clarity and can reduce frustration of the user. This is the most important functionality that would not only greatly improve this specific problem, but the HFL in general, as you get more of a conversation, a back and forth of user and LLM, which increases clarity for both parties. Secondly, more variables can be given to the LLM. If ChatGPT did have access to the length of the level, then it would also fix the problem of it not doing anything in that specific case. This approach however takes a lot of effort since the functionality needs to be added to the code and it needs several evaluations to correctly prompt the explanation of each variable. If developing this concept for an actual user application, then this becomes one of the most important and time-consuming tasks. Thirdly, users in the user evaluation mentioned they would like some example prompts. By informing the users about what they can change in the level, it will happen less often that the user asks for something the application cannot do. Combining this with the first improvement, the LLM can play a role in this process, as it can give a set of example prompts either at the start or as a response to a request it cannot fulfil. Lastly, in the case of the LLM adding a new variable, this problem can be easily fixed by better prompting. If the start explanation prompt tells the LLM not to create new variables, then it will likely not do that.

In case of the prototype doing something wrong, since this is a problem about understanding the variables and how they are used, this is mostly a problem about prompting. There is only 1 solution to this, which is better explaining the variables. If for example the explanation of the variable "SpawnEnemy" specifically mentions that if the user wants no enemies, a very high number should be provided, then it will never make the mistake again of setting the value to 0 doing the opposite of what the user wants. This can become a very tedious task to do however when you have a lot of variables to change, and a lot of things that can go wrong for each variable. Fixing this requires a lot of testing and evaluations to find out all the possible things it can do wrong and find out what phrasing will fix it. Ideally, the LLM understands completely how the whole level is generated, where in the code each variable is used and how it impacts the output simply by providing the code. Current LLMs unfortunately are not advanced enough yet to do this, but with the rapid advancements in understanding and generating code, it might not take long for LLMs to be able to do this, even for very complicated applications. Although LLMs cannot understand everything with just code yet, this situation might be improved by explaining how the levels are generated in pseudo-code. It is not sure yet if this would work, but it would be interesting for future research.

6.2 Concept Applications

This section focuses on how this concept can be best utilised and why. During the user evaluations, participants mentioned they could see the concept work for both level designers and users. This concept has a lot of potential as a tool for level designers. The ability to translate human text to elements in a level makes the design process faster and easier to use. This concept shows a lot of promise either as a way to implement changes faster in computer-aided MIPCG, or by letting the computer generate the levels and the designer change specific parts of the level in a level editor. By applying this concept, level designers can quickly reach a configuration for certain types of levels it envisions, generate a lot of levels with that configuration and finish them up in the editor. This way the tool can be used to speed up the process.

By giving this tool to users, they can let the computer generate levels they prefer. In some sense, the users become level designers. Implementing this concept in an existing game and giving the users direct access to this tool means they can either generate these levels, share them with others and compare how well they do, or the game essentially turns into a "sandbox" game, where they have complete freedom and they can satisfy certain fantasies they might have in the game. Turning this concept into a game mechanic for existing games brings its challenges, however. One can imagine that granting the power to change the difficulty of the game at a whim to the user means giving meaning to actually passing the level very difficult. If a user can remove all enemies and gaps, then it becomes impossible for the player to die. Passing the level is not an achievement anymore. This means the concept should not be used as a game mechanic in single-player games, or else winning the game loses its meaning and the game becomes a "sandbox" game. However, any game where you can compete against other players playing the same level shows the most potential of utilising the strengths of this concept directly. Since the levels that are played do not have to be "balanced", it is not a failure case if the game generates a very hard or very easy level. The only thing that matters is if the user enjoys playing the level. This means that most games where players play on the same track or level, and where the goal is to complete the track or level as fast as possible, are potentially well suited for this concept.

This concept can also be applied to different game elements than level generation. The more impact the game element has on the gameplay, the more challenging it is to apply this concept as a game mechanic, and the more useful it becomes to use it as a tool. Since the level has a lot of impact on how the game is played, it will be hard to balance the level as mentioned before, but if you apply this concept to a game element that has less impact on the game, it can be more easily balanced and can be a great addition. For example, applying this to the generation of weapons in a game, it can be interesting to let the user decide if it wants more damage, speed, ammunition or another aspect of the weapon, and balancing the generated weapon will be much easier than balancing, for example, an entire world or level. This is also the reason why this concept can work well on the aesthetics of the game, as it has no impact on the gameplay and it does need to be balanced.

Some users showed interest in changing the rules and mechanics of the game, such as getting the ability to fly in a game where you should only be allowed to jump. Implementing features like this will be very challenging. For this idea to work, LLMs need to become powerful enough to understand the whole code behind a game and be able to generate code that accurately reflects what the user envisioned. The LLM will have full power to change most of the, if not all the code in the game. If the LLM is not capable enough, it has the potential to break the game, making levels completely unplayable. This would ruin the user experience. However, when the LLM is capable enough, it opens up possibilities for users to completely design levels, even whole games from scratch only with text. This would be a huge step in the game industry and might give rise to a whole new generation of games. The more powerful the LLM, the more power over the game you can entrust it with, and the more configurable the game becomes with just text. Although this is far from reality yet, with the current advancements in the field of AI and LLM, someday in the (maybe not so far) future, we might see this become possible.

For this concept to function best, it is important to encourage users to utilize the feedback loop, as the program learns the preferences of the user through many iterations. From the user testing, it became evident that participants focus a lot on the gameplay and less on generating better levels. This can have several causes, some are discussed here. Firstly, it can be that the levels are already "fun" and "good", in this case, there is no need to increase the use of the feedback loop. Secondly, it is possible the users are not aware of the things they can change, they might not think about the option of improving the level through the feedback function, even though it has clearly been stated at the beginning. Thirdly, the function may look dull and unattractive to interact with, which dissuades the users from using it. Lastly, it can be the case that users feel like they first have to complete a level before they "deserve" to give it feedback. Most of these cases have clear solutions to improve the situation, like specifically notifying the user of what it can do and when, or integrating this function more into the game. However, before trying to improve this, it is important to understand why users behave this way. For this more user tests have to be conducted.

6.3 Reflection

This section reflects on the process and choices made during the project.

The idea for this project sparked from a passion for gaming and an interest in programming, and while the project was a clear intersection of these 2 fields, it also had a lot of emphasis on web development. The idea of the project initially was to research how to harness the power of LLMs in PCG. From this idea, the concept of an HFL sparked forth, which showed promise to improve PCG. The first plan was to apply this concept to the study of MarioGPT. However, after trying to understand the study and finding out if incorporating the concept is possible, it became apparent that it was hard to change the output of the level generator in an explainable way. As mentioned before, this can be a challenge when working with Machine Learning, as it is very hard to understand how the model gets to the output with the given input, it is like a "black box". This does not always pose a problem though. In this project, ChatGPT is also a model that uses machine learning, it is also a "black box". However, when using this program, the output is somewhat predictable with the given input, and the program produces reliable results. When using ChatGPT for this project, there was rarely a failure case caused by the unreliability of ChatGPT, and these failures were mostly fixed by changing the input. Of course, ChatGPT can be unreliable at times, but these cases are sporadic and can be built around. The rate at which these unreliabilities happen does not justify not using it, as the benefits far outweigh the risks. In the case of MarioGPT, it was too unreliable in use, which made it unsuited for testing the concept of an HFL.

The second plan was to apply the concept to a simpler form of PCG, namely constructive PCG. The code was easy to grasp and was well-suited for testing the concept. It was however a project that is written in Javascript and HTML. This causes the project to have a lot of emphasis on web development, more than anticipated. Since the main developer of this project was not experienced in the world of web development, considerable time was spent on typical problems within this field, that could have been easily fixed by someone who is experienced with web development. This time instead could have been spent on improving the prototype and conducting more user evaluations after these improvements. The prototype however can provide an answer to the research question and prove the concept.

The initial plan, building upon the MarioGPT study, meant there was more of a focus on computer-generated Mixed-Initiative PCG, as the idea was to let the program provide a few levels to the user and the user indicating its preferences. This meant there was limited human input, causing most of the output to be decided by the computer. However, changing to a simpler PCG technique caused the project to shift more to computer-aided MIPCG. Since the user can provide text as input, it has more power over the outcome of the level. This means the user more actively thinks about what it wants in the level, causing the Mixed-Initiative system to shift more power to the user. It also became evident that this concept can prove to work well as a tool to design levels. In that case, it shifts even more to computer-aided MIPCG. The prototype as it is now, is somewhere close to the middle of the 2 spectrums, given that the user uses the feedback function.

7 Conclusion

How can the integration of a Human Feedback Loop using LLMs improve PCG techniques in level design for games? Generated content through PCG can be improved by learning the preferences of the user through text using an HFL and ChatGPT, allowing the algorithm to create more personalized levels. This is proven by analysing the current body of knowledge, devising a plan that can answer the question, carrying out that plan and conducting user evaluations to test it. The plan was to realise a prototype of the concept that is built upon a Super Mario Bros level generator. Using the interpretive capabilities of ChatGPT, user preferences in the form of text can be translated to elements within the level, granting a wide range of users the power to accessibly change the level to their liking.

To analyse the concept and the prototype, user evaluations were conducted, providing useful insights into the flaws and strengths of the concept. An important improvement for the functionality of the prototype and the concept is to allow the LLM to talk back to the user, instead of only providing the settings for a new level, creating more space for a back-and-forth between the user and the computer. This also allows the LLM to explain its choices when it cannot fulfil the demands of the user, reducing user frustration. When improving the functionality of the prototype, most issues can be fixed by changing the explanation prompt for the LLM. Finding out what needs to be changed and how to change the explanation takes a lot of testing and can be a time-consuming process.

This concept shows the most promise as a tool for level design, either for game developers or players. Allowing users to use text to change specific elements in the level improves the efficiency, speed and accessibility when designing a level, which can greatly improve the process of designing levels. This makes this concept well-suited for computer-aided MIPCG and games where players play the same level, for example, racing games. Implementing this concept for existing games as a game mechanic however brings its challenges and should mostly be applied to changing game elements that have little to no impact on gameplay such as aesthetics, as balancing the output can be difficult.

Applying this concept to Reinforced Learning came with challenges. Since unexplainable processes happen within this PCG technique, if the output is not as desired, it will be hard to locate the problem. As this research failed to apply the concept to RL, the PCG technique and therefore the shape of the HFL had to be changed. A simpler PCG technique and approach for the HFL was chosen to prove the concept. This also caused the project to shift more towards web development, which slowed down the process of development due to the lack of experience of the researcher in the field of web development.

Current LLMs are powerful enough to put this concept to use in different contexts, but as the capabilities of LLMs increase, their applications and effectiveness within the game industry will also expand. The more the LLM can be aware of the context with minimal information and still provide desirable output, the more control over elements in the game can be given to LLMs. This causes the designing process to become even more efficient and creates even use cases within the game industry.

This concept proved to work for level design in the game Super Mario Bros, which uses constructive PCG, but there are more opportunities for research with this concept. Research involving other PCG techniques, game elements or the generation of content for other games is still left undiscovered, potentially hiding even more promising use cases.

To conclude, integrating LLMs into PCG can help generate personalized game content, providing an accessible and efficient solution in the process of level design. It can be used as a powerful tool in level design or when applied in the right context as a game mechanic. This research shows the world a glimpse of the potential of LLMs in game design since there are still improvements and undiscovered researched opportunities that have the potential to show even more use cases for the concept.

8 Recommendations

This section provides some suggestions for future research. As stated before, the emergence of LLMs is very recent and LLMs have many undiscovered applications. This study only researches a small, very specific application of an LLM within the game industry, but there are so many more potential use cases. As there are way too many possible applications, this section only covers the recommendations for applications of the HFL using LLMs within the game industry.

Firstly, this study only tested the concept on Super Mario Bros. There are so many more potential games that can benefit from the concept. Applying this concept to the Track Generator of Track Mania Turbo would be very interesting, as the strengths of the program can be utilised very well within that framework and the game is mostly open source, but any racing game can work well to test this concept. This can also test the performance of the concept within 3D games. Puzzle games can also show potential, as the levels can be played by other players and speed of completion can be measured and compared. The PCG techniques behind generating levels might however not be well suited for integrating this concept directly and can potentially be challenging. Other 2D platform games are also possible candidates for performing future research. Using the concept as a level design tool, its capabilities can also be well used in designing maps or levels in strategy games like Civilization 5 (2010) or XCOM (2020).

Secondly, this study researched the application of the concept on constructive PCG and touched upon applying it to Machine Learning. There are however many more PCG techniques that can potentially benefit from an HFL. As mentioned in the background research, reinforced learning, supervised learning and potentially solver-based systems or general advisory networks can be interesting PCG techniques to test this concept. Evolutionary algorithms is an area within PCG that already researched the usage of an HFL to improve the selection phase of the evolutionary cycle. However, there has been no research on the usage of a LLM within the system. This application can also be potentially interesting for future research.

Thirdly, this study researched the concept on level design. There are however many more elements in the game that this can be applied to. Think about the generation of worlds, narratives, weapons, enemies, cards, special powers etc. Anything that can be procedurally generated has the potential to use LLMs to improve the generation process. As mentioned before, a game element that can benefit greatly from the concept is the aesthetics of a game. Think about backgrounds, colours of characters, enemies, game objects, shapes and movements of things in the game. Even changing the parameters of procedural animation can be very interesting for the user, and none of these elements suffer from reducing playability, except maybe the visibility of certain things.

Lastly, using the concept as a game mechanic and building a game around the idea that the user can interact with the game through text can be very interesting. A game that already explored this is façade(2005), but they only used it for narratives. Imagine a game where the user can shape the landscape, or generate objects, and use these elements to complete the game. It allows for a game where creativity and wittiness can become your tools for completing the game. There are many shapes a game like this can take, and as the capabilities of LLMs improve, developing games like this can become more and more interesting.

References

[1] M. Dahrén, "The Usage of PCG Techniques Within Different Game Genres," Tech. Rep., 2021.

[2] N. Shaker, J. Togelius, and M. J. Nelson, "Computational Synthesis and Creative Systems Procedural Content Generation in Games," Tech. Rep., 2016. [Online]. Available: http://www.springer.com/series/15219

[3] N. N. Blackburn, M. Gardone, and D. S. Brown, "Player-Centric Procedural Content Generation: Enhancing Runtime Customization by Integrating Real-Time Player Feedback," in *CHI PLAY 2023 - Companion Proceedings of the Annual Symposium on Computer-Human Interaction in Play*. Association for Computing Machinery, Inc, 10 2023, pp. 10–16.

[4] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," in *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, 9 2011, pp. 172–186.

[5] G. N. Yannakakis and J. Togelius, "Artificial Intelligence and Games," Tech. Rep., 2018.

[6] W. L. Raffe, F. Zambetta, X. Li, and K. O. Stanley, "Integrated Approach to Personalized Procedural Map Generation Using Evolutionary Algorithms," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, no. 2, pp. 139–155, 6 2015.

[7] P. T. Ølsted, B. Ma, and S. Risi, *Interactive Evolution of Levels for a Competitive Multiplayer FPS*, 2015.

[8] S. P. Walton, A. A. M. Rahat, and J. Stovold, "Evaluating Mixed-Initiative Procedural Level Design Tools using a Triple-Blind Mixed-Method User Study," 5 2020. [Online]. Available: http://arxiv.org/abs/2005.07478 http://dx.doi.org/10.1109/TG.2021.3086215

[9] W. L. Raffe, F. Zambetta, and X. Li, *Neuroevolution of Content Layout in the PCG: Angry Bots Video Game*, 2013.

[10] P. L. Lanzi, ACM Digital Library., and Association for Computing Machinery. SIGEVO., *Proceedings of the 13th annual conference on Genetic and evolutionary computation.* ACM, 2011.

[11] D. Gravina, A. Khalifa, A. Liapis, J. Togelius, and G. N. Yannakakis, *Procedural Content Generation through Quality Diversity*, 2019.

[12] A. Summerville, S. Snodgrass, M. Guzdial, C. Holmgård, A. K. Hoover, A. Isaksen, A. Nealen, and J. Togelius, "Procedural content generation via machine learning (PCGML)," *IEEE Transactions on Games*, vol. 10, no. 3, pp. 257–270, 9 2018.

[13] M. Wiering and M. van Otterlo, *Reinforcement Learning, State-of-the-Art*, 2012.

[14] A. Khalifa, P. Bontrager, S. Earle, and J. Togelius, "PCGRL: Procedural Content Generation via Reinforcement Learning," Tech. Rep., 2020. [Online]. Available: www.aaai.org

[15] P. Cunningham, M. Cord, and S. J. Delany, "Supervised learning," in *Cognitive Technologies*. Springer Verlag, 2008, pp. 21–49.

[16] M. Awiszus, F. Schubert, and B. Rosenhahn, "Wor(l)d-GAN: Toward Natural-Language-Based PCG in Minecraft," *IEEE Transactions on Games*, vol. 15, no. 2, pp. 182–192, 6 2023.

[17] G. Smith, J. Whitehead, and M. Mateas, "Tanagra: Reactive planning and constraint solving for mixed-initiative level design," in *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, 9 2011, pp. 201–215.

[18] B. D. Spelchan, "A Deep Learning Based Mixed Initiative Editor for Game Level Generation," Tech. Rep., 2022.

[19] M. Guzdial, N. Liao, and M. Riedl, "Co-Creative Level Design via Machine Learning," 9 2018. [Online]. Available: http://arxiv.org/abs/1809.09420

[20] O. Delarosa, H. Dong, M. Ruan, A. Khalifa, and J. Togelius, "Mixed-Initiative Level Design with RL Brush," 8 2020. [Online]. Available: http://arxiv.org/abs/2008.02778

[21] T. Shu, J. Liu, and G. N. Yannakakis, "Experience-Driven PCG via Reinforcement Learning: A Super Mario Bros Study," in *IEEE Conference on Computatonal Intelligence and Games, CIG*, vol. 2021-August. IEEE Computer Society, 2021.

[22] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," *IEEE Transactions on Affective Computing*, vol. 2, no. 3, pp. 147–161, 7 2011.

[23] M. U. Hadi, R. Qureshi, A. Shah, M. Irfan, A. Zafar, M. B. Shaikh, N. Akhtar, J. Wu, S. Mirjalili *et al.*, "A survey on large language models: Applications, challenges, limitations, and practical usage," *Authorea Preprints*, 2023.

[24] X. Peng, J. Quaye, W. Xu, C. Brockett, B. Dolan, N. Jojic, G. DesGarennes, K. Lobb, M. Xu, J. Leandro, C. Jin, and S. Rao, "Player-Driven Emergence in LLM-Driven Game Narrative," 4 2024. [Online]. Available: http://arxiv.org/abs/2404.17027

[25] Y. Sun, H. Wang, P. M. Chan, M. Tabibi, Y. Zhang, H. Lu, Y. Chen, C. H. Lee, and A. Asadipour, "Fictional worlds, real connections: Developing community storytelling social chatbots through llms," *arXiv preprint arXiv:2309.11478*, 2023.

[26] Q. Guignard, "Exploring llms and mcts for emergent narrative," Master's thesis, ETH Zurich, 2024.

[27] G. Todd, S. Earle, M. U. Nasir, M. C. Green, and J. Togelius, "Level Generation Through Large Language Models," 2 2023. [Online]. Available: http://arxiv.org/abs/2302.05817 http://dx.doi.org/10.1145/3582437.3587211

[28] S. Sudhakaran, M. González-Duque, C. Glanois, M. Freiberger, E. Najarro, and S. Risi, "MarioGPT: Open-Ended Text2Level Generation through Large Language Models," 2 2023. [Online]. Available: http://arxiv.org/abs/2302.05981

[29] "Super Mario Level Generator MarioHTML5." [Online]. Available: https://github.com/robertkleffner/mariohtml5

[30] "Super Mario Level Generator MarioAI." [Online]. Available: https://github.com/mcgreentn/MarioAI

[31] "Super Mario Level Generator Infinite-tux." [Online]. Available: https://github.com/qbancoffee/infinite-tux

[32] "Skypack OpenAI package link." [Online]. Available: https://cdn.skypack.dev/openai

[33] "Skypack Seedrandom package link." [Online]. Available: https://www.skypack.dev/view/seedrandom

9 Appendix

```
STATISTICS = {
    "enemy": np.array([1.0, 3.0, 7.0]),  # 1, 3, 7
    "pipe": np.array([1.0, 2.0, 5.0]),  # 1, 2, 5
    "block": np.array([500.0, 750.0, 1760.0]),  # 50, 75, 176
}

d["enemy"] = stats.mstats.mquantiles(enemy_counts, [0.33, 0.66, 0.95])  # 0.33, 0.66, 0.95
d["pipe"] = stats.mstats.mquantiles(pipe_counts, [0.33, 0.66, 0.95])  # 0.33, 0.66, 0.95
d["block"] = stats.mstats.mquantiles(block_counts, [0.33, 0.66, 0.95])  # 0.33, 0.66, 0.95

prompts = ["many blocks, no pipes"]

# generate level of size 1400, pump
generated_level = mario_lm.sample(
    prompts=prompts,
    num_steps=420,
    temperature=2.0,
    use_tqdm=True
```

Figure 10: Input A (the numbers after the hashtag are the values of the original code)

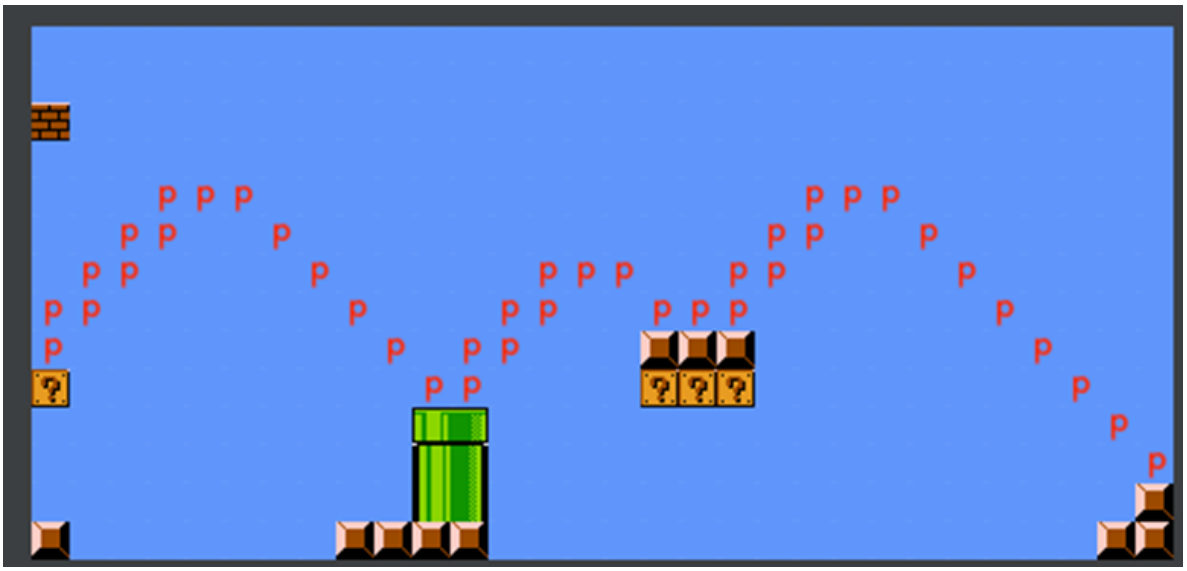

Figure 11: Generated level with input A

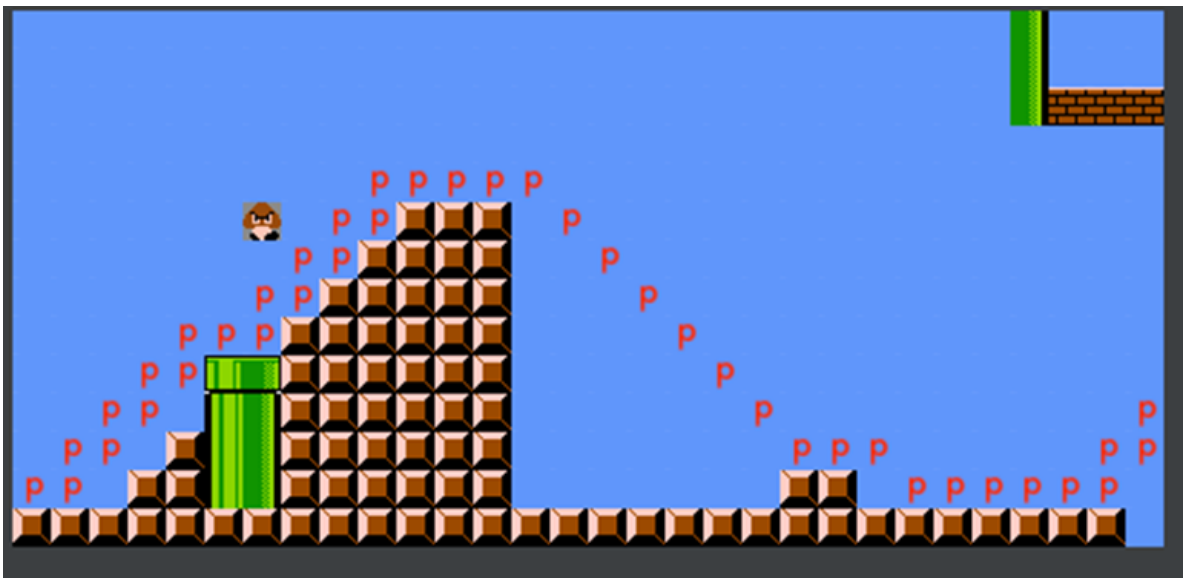

Figure 12: Generated Level with Input A

```
STATISTICS = {
    "enemy": np.array([1.0, 3.0, 7.0]),  # 1, 3, 7
    "pipe": np.array([1.0, 2.0, 5.0]),  # 1, 2, 5
    "block": np.array([50.0, 70.0, 176.0]),  # 50, 75, 176
}
d["enemy"] = stats.mstats.mquantiles(enemy_counts, [0.33, 0.66, 0.95])  # 0.33, 0.66, 0.95
d["pipe"] = stats.mstats.mquantiles(pipe_counts, [0.93, 0.96, 0.95])  # 0.33, 0.66, 0.95
d["block"] = stats.mstats.mquantiles(block_counts, [0.93, 0.96, 0.95])  # 0.33, 0.66, 0.95

prompts = ["many blocks, many pipes"]

# generate level of size 1400, pump t
generated_level = mario_lm.sample(
    prompts=prompts,
    num_steps=420,
    temperature=2.0,
    use_tqdm=True
)
```

Figure 13: Input B

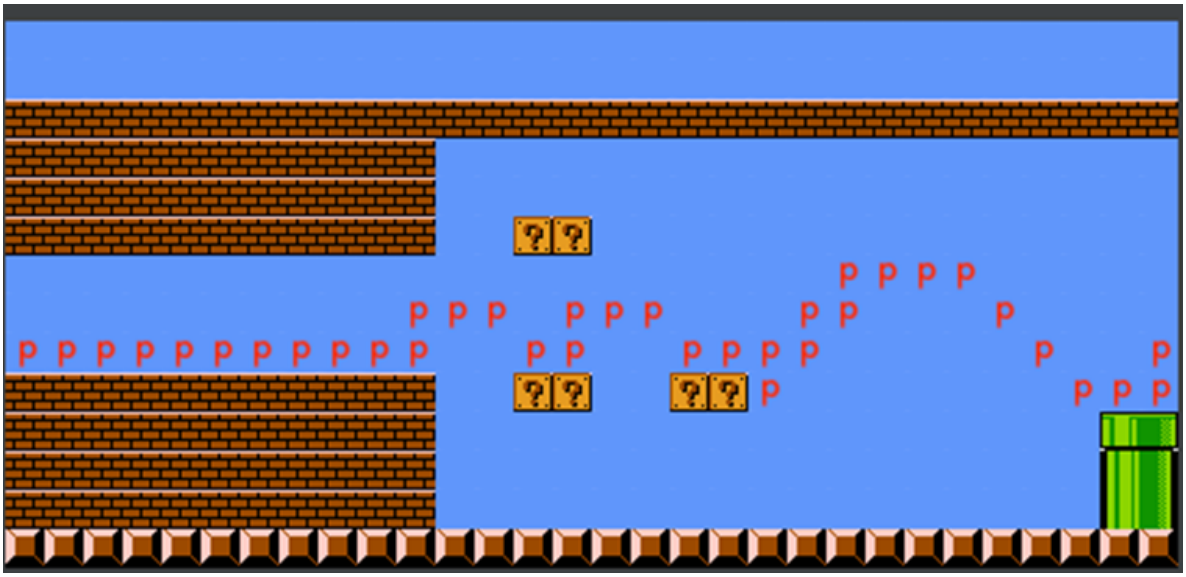

Figure 14: Generated level with input B

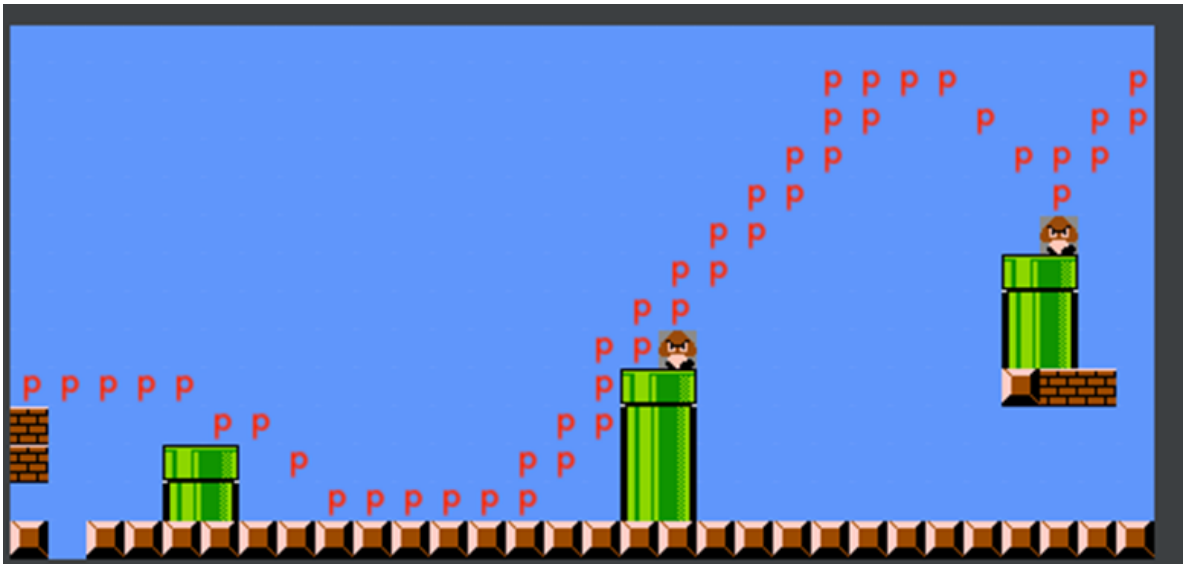

Figure 15: Generated level with input B

```
STATISTICS = {
    "enemy": np.array([1.0, 3.0, 7.0]),  # 1, 3, 7
    "pipe": np.array([1.0, 2.0, 5.0]),  # 1, 2, 5
    "block": np.array([5.0, 7.0, 10.0]),  # 50, 75, 176
}
d["enemy"] = stats.mstats.mquantiles(enemy_counts, [0.33, 0.66, 0.95])  # 0.33, 0.66, 0.95
d["pipe"] = stats.mstats.mquantiles(pipe_counts, [0.33, 0.66, 0.95])  # 0.33, 0.66, 0.95
d["block"] = stats.mstats.mquantiles(block_counts, [0.33, 0.66, 0.95])  # 0.33, 0.66, 0.95
```

```
prompts = ["many blocks, no pipes"]

# generate level of size 1400, pump
generated_level = mario_lm.sample(
    prompts=prompts,
    num_steps=420,
    temperature=2.0,
    use_tqdm=True
```

Figure 16: Input C

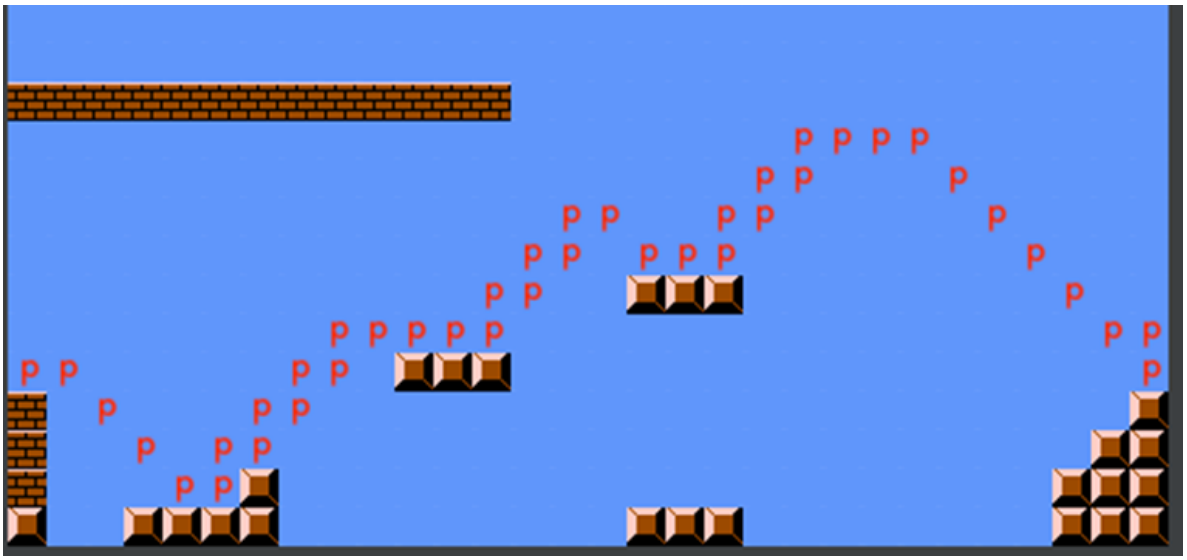

Figure 17: Generated level with input C

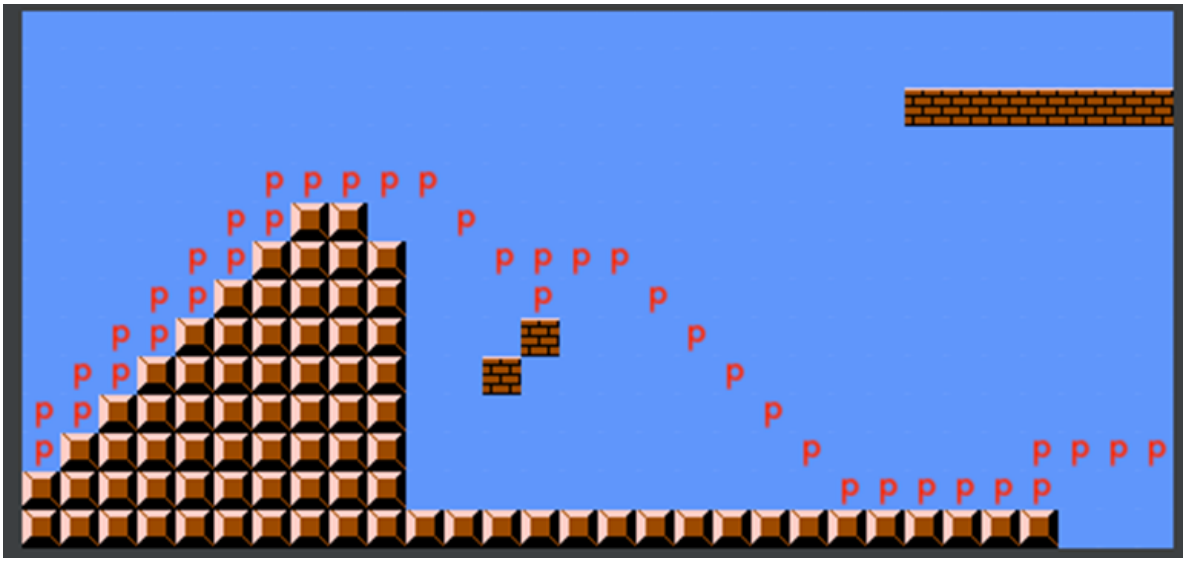

Figure 18: Generated level with input C

```
  "enemy": np.array([1.0, 3.0, 7.0]),  # 1, 3, 7
  "pipe": np.array([1.0, 2.0, 5.0]),  # 1, 2, 5
  "block": np.array([5.0, 7.0, 10.0]),  # 50, 75, 176
```

```
d["enemy"] = stats.mstats.mquantiles(enemy_counts, [0.33, 0.66, 0.95])  # 0.33, 0.66, 0.95
d["pipe"] = stats.mstats.mquantiles(pipe_counts, [0.33, 0.66, 0.95])  # 0.33, 0.66, 0.95
d["block"] = stats.mstats.mquantiles(block_counts, [0.1, 0.2, 0.3])  # 0.33, 0.66, 0.95
```

```
prompts = ["many blocks, no pipes"]

# generate level of size 1400, pump
generated_level = mario_lm.sample(
    prompts=prompts,
    num_steps=420,
    temperature=2.4,
    use_tqdm=True
```

Figure 19: Input D

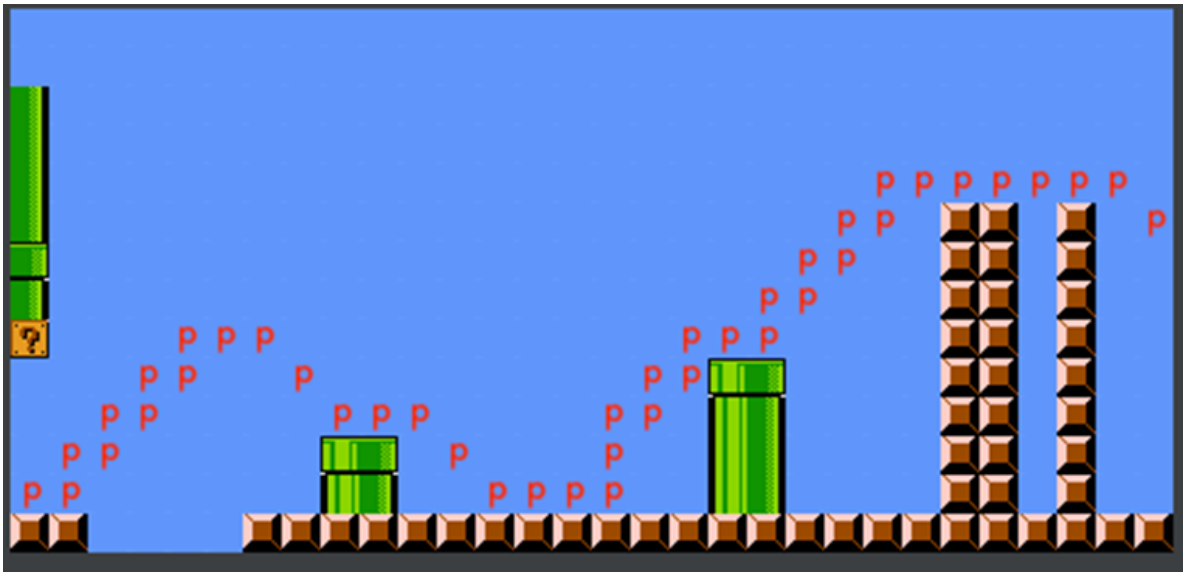

Figure 20: Generated Level with input D

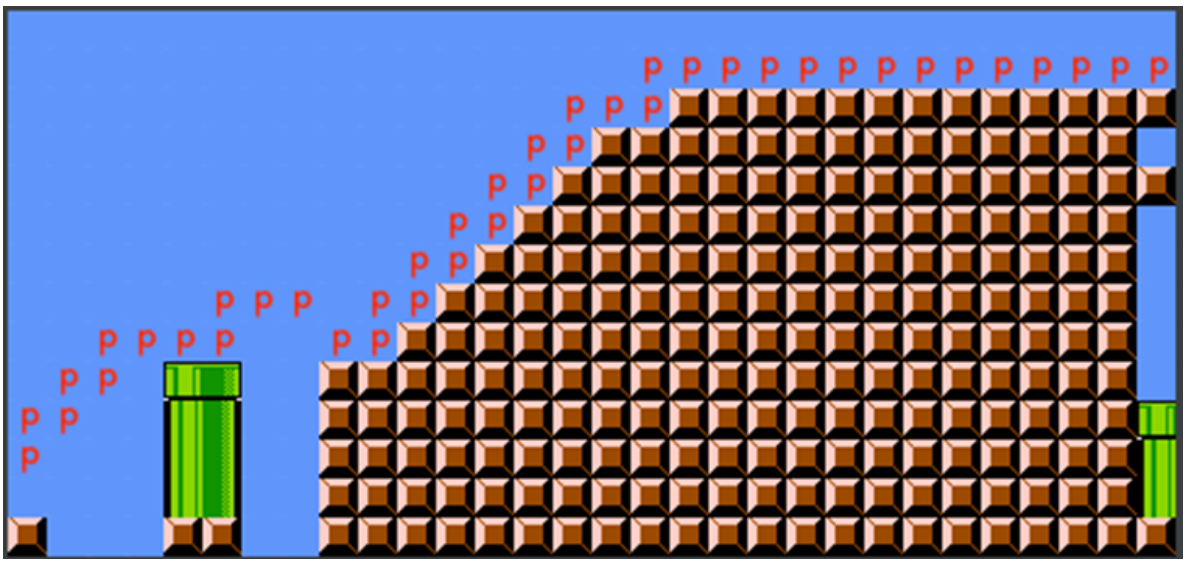

Figure 21: Generated Level input D