Generating Procedural Dungeons Using Machine Learning Methods

Mariana Werneck
Department of Computer Science
Universidade Federal Fluminense
Niterói, Brazil
marianawerneck@id.uff.br

Esteban W. G. Clua
Department of Computer Science
Universidade Federal Fluminense
Niterói, Brazil
esteban@ic.uff.br

***Abstract*—Procedural content generation (*PCG*) is a powerful tool to optimize creation of content in the game industry. However, it can lead to lack of control and mischaracterization of the game design, creating unbalanced or undesired situations. To overcome such problems, machine learning can be used to map important patterns of a game design and apply them in the PCG. Considering such aspects, this paper proposes a strategy for procedurally generating dungeons using ML techniques. We use Unity ML-Agents tool for the implementation, since dungeons are environments largely used in the industry that also require more control over its creation. The strategy used in this paper has proven to generate dungeons that respect room positioning design choices and maintains the game characterization. We conclude, after conducting a survey with users, that the generated dungeons presented reliable maps and showed to be more enjoyable and replayable than manually generated ones following the same design principles.**

Keywords—procedural generation, Machine Learning, Dungeons, UnityML

I. INTRODUCTION

The video game industry represents nowadays, a billionaire market with a revenue of more than $100 billion annually [1]. Due to this market being so lucrative and ludic, more and more developers join the independent segment, even though the development costs have been raising through time [2]. As an alternative to lower game production costs, free game engines and asset stores have been taking the scene in the last years, but even though more resources are available for independent game production, the activity is still costly and time-consuming.

Considering this, the procedural content generation (PCG), which is the creation of content using an algorithm, has the potential to become the key point for a more fluid and sustainable game development. The main points that makes PCG so powerful is that it allows many improvements in the process of game creation, such as increasing the amount of diversity in the content of a game, creating content more rapidly and, thus, reducing costs and time in game production [3].

Since PCG can represent such advantage to the game development process, it has always found its way to be used in the game industry. In 1980, the game Rogue, developed by Michael Toy and Glenn Wichman at the University of California became the first popular game with randomly generated levels, population of monsters and treasures [6], using pseudorandom number generators strategies [5]. Its popularity made roguelike the term to designate games with randomly generated dungeons.

With the expansion of the game market between 1990s and 2000s, a number of role-playing games inspired by Rogue emerged. One of the most successful among them was Blizzard's Diablo franchise, that had premade map tiles being randomly chosen and aggregated to form dungeon levels. More recently, Minecraft has made huge success using procedural algorithms to generate an open world environment.

However, when using procedural content generation, it is easy to face problems such as lack of control and repeated patterns over the created objects and scenarios, which can lead to mischaracterization of the design of the game, or lack of diversity in the generation. These problems can make game designers refrain from the use of PCG [3].

Focusing on the procedural generation of levels, this work aims to present a strategy for procedural generation of dungeons, following the dungeon crawler game style, where the player navigates through a labyrinth of rooms, organized in levels, to fulfill an objective like save himself or a princess, and has to defeat enemies or escape traps along the way. The player also can find items and power ups along the way, which contributes to the game progression. The display of the rooms of the labyrinth is also important for the experience of the game, since the rooms are often diversified in terms of what challenges or rewards a player can find on them, being categorized this way. The patterns in which the special rooms appear on a dungeon tend to repeat on its levels [3].

The chosen genre of scenarios has been used in extremely successful games like The Legend of Zelda, Diablo 2, and more recently, The Binding of Isaac, and have proven to be able to adapt well with the diversity of content, contributing to these games' ability to be played many times for the same player. This characteristic is known as replayability and is very important for the long-term enjoyment of video games and is a factor that procedural generation of content can add the most [5]. Also, it is important for procedural content to follow patterns that deliver replayability because, this way, the generation algorithm can be responsible for generating a wider range of content without needing further adjustment. In addition, dungeons are an environment that needs more control over its creation, because its design longs to have a more controlled progress for the players and can suffer from too much randomness. Considering this, automated dungeon generation has similar challenges to its manual generation [3].

The proposed strategy has the objective of procedurally generating credible and reliable dungeons that preserve the sense of belonging to a unique game. The use of machine learning methods in the generation has the purpose of making the generation more adaptive to a wider range of game and level designs. For the use of machine learning methods, this work integrates Unity ML-Agents, which is a promising and more ready-to-use library for using machine learning to train agents developed by Unity Technologies and that is recently available for use in the Unity game engine [11]. Thus, being Unity a game engine widely used for independent game and studios developers, the proposed algorithm aims to aggregate as a strategy for procedural dungeon generation tangible for the independent game creators.

II. Related Works

The procedural generation of levels has been approached in many ways in the past since it represents an effective way to lower game production costs and time. Of the known ways of generating procedural dungeons, one traditional and successfully used approach was made by Johnson et al. using the cellular automata [8]. In their method, they begin with a 50 x 50 grid as the basis of their cellular automata. After generating this first grid, other grids are generated at the cardinal points of the initial grid with the purpose of expanding the map. Then, each cell contained on the grid is fulfilled with information about its position within the grid, the state of the neighborhood cells, the type of the cells (rock, floor or wall) and the cell's group number. With the objective of creating a map shape in the grid, a rule set is defined to iterate over the grid. The rule set defined by Johnson et al. stated that: 1) a cell is rock if the neighborhood value is greater than or equal to 5 and floor otherwise; and 2) a rock cell that has a neighboring floor cell is a wall cell. This rule set is applied iteratively until shapes begin to form in the grid, usually, these shapes tend to look like caves or forest labyrinths, as it is shown in Fig. 1.

The positive aspects of this method is that it can generate levels with efficiency, being able to generate levels in runtime, and thus is capable of generating infinite levels. However, the cellular automata method lacks control in the generation process, being hard to predict the results of the generation, and it needs further treatment to guarantee that the player will not get stuck in a floor area that does not connect to others. For the chaotic aspect of the generated levels, this generation method is more likely to be used in games such as cave crawler.

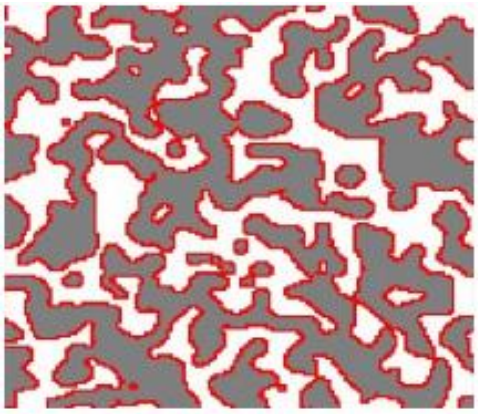

Fig. 1. Gray areas are categorized as floor, red and white represent rock and wall respectively in this map generated with the cellular automata [8].

Another interesting successful approach for dungeon generations is the space partitioning method described by Shaker et al. [9]. In this work, the space selected to the map is divided recursively using a partition technique, in Shaker et al. work, the chosen partition technique was the binary space partition. The advantages of this method is that the partitioned space can be represented as a binary tree, also known as BSP tree. This offer some advantages as navigation and manipulation of the generated map, since binary trees are widely known and studied, and therefore there is a wide range of optimized algorithms that deal with binary tree manipulation and search problems. Because of this aspect, the generated level can vary according to the implemented algorithm to generated the BSP variant. Multiples algorithms can be adjusted according to the game designer needs for the generated levels.

After the space is partitioned, a room is placed inside each partition. This is done by randomly selecting room corners and verifying if the chosen space is acceptable for a room area, for example, testing if is not too small or if it has the shape of a square. Then, corridors are placed by connecting children of the same parent to each other. An example of the resulted dungeons is shown in Fig. 2.

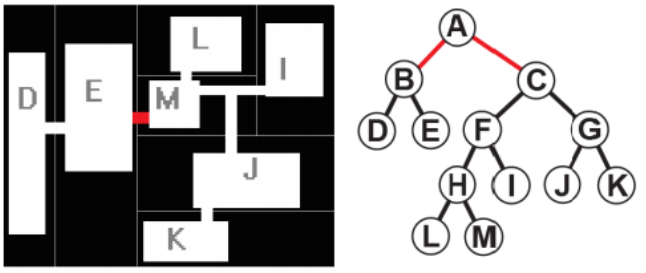

Fig. 2. A dungeon map generated by BSP special partitioning method. The white areas represents rooms and corridors, the black ones represent walls and rocks. The red represents the creation of a corridor [9].

Shaker et al. also describes a way of generating dungeons using agents, which he calls agent-based dungeon growing. In this method the agent is generated in a random tile through a tile grid representing the dungeon map, and then it begins to digger through the grid, with a property that determines the chances of spawning a room on that way. The probability of the agent spawning a room increases as the agents get farther away from the beginning. After a room is spawned, the property is set to zero once more, and the beginning point becomes the last spawned room coordinate [9].

The agent parameters used by Shaker et al. were simple; it had a blind digger agent wandering in the dungeon space, divided in grids. At every grid navigated by the agent, the probability of the agent choosing to spawn a room increased 5%. Every time a room was spawned the probability was reduced to 5% again. The probability for the agent making a turn was around 25% to 30%.

This method depends on the intelligence of the agent for choosing to dig more interesting paths, therefore forming better dungeons. When using a blind digger agent, the dungeons turn out to be too random and mischaracterized, which turns this method less likely to be used since the game designer may lose control of the result when the agent is not well controlled. This can result in poorly developed dungeons, since dungeons are a more sensitive environment in terms of room arrangement and player experience. In addition, it requires treatment to prevent generated rooms from overlapping each other.

There are also some contributions for the generation of levels using machine-learning algorithms, but most of them focus on platform levels and reproduction of levels of iconic games, such as Mario, known for the great design of their levels. Among these works, one that is interesting and relevant is Dahlskog and Togelius work in which they propose the usage of patterns, categorized as micro, meso and macro-patterns, to generate Super Mario Bros levels [10]. The patterns are slices of Super Mario Bros platform levels, and they use an evolutionary algorithm to recombine these slices, forming new Super Mario levels this way. The used

evolutionary algorithm has three different fitness functions to evaluate the generated levels.

In their method, they noted that the used slice patterns in the methods can affect the quantity and way that other patterns can appear, what can result in lack of diversity of the generated levels. The macro patterns were the ones that presented more large-structure in the platform level, but they were also the patterns that took the longest to generate a dungeon. However, their method showed some efficiency considering the time of the level generation when using micro and meso patterns, although these patterns generated more chaotic results.

III. PROCEDURAL DUNGEON GENERATION

The dungeon generation method proposed on this paper uses agents to create the room paths and decide on the placement of the dungeons strategic rooms. Since procedural level generation using agents tends to have results proportional to the intelligence of the generating agent, this method uses machine-learning methods available at Unity ML-Agents library.

The proposed method can be divided in two big phases: 1) The design of the dungeon general structure and 2) The training and implementation of the agents. The design of the dungeon general structure is the phase where we are looking for ways to delineate what our dungeon levels should look like, and so, define based on what initial configuration the agents should be trained so that they can deliver dungeons similar to what we aspire.

In this first phase, the most important design aspect to be considered in the presented method is the relative configuration of the dungeon strategic rooms. In dungeon crawler games, the events that are usually held at different and specific rooms, such as boss fight and equipment acquisition, like the shop where the player can buy items or a treasure room where the player receive items as a reward, give the player the sense of conquest progression in the level. These types of rooms should be placed very strategically in the dungeon and they will be referred in this paper as event rooms.

To begin the design of the dungeon structure, it is important to decide what kind of event rooms are better suited for the game and level design. It is also important to define characteristics of the positioning of the rooms that will hold the events because they will guide the progression of the levels. In this paper, the chosen event rooms were the boss room, which holds boss fights, shop rooms that represent equipment acquisition through purchase with money acquired in the game, and the treasure room, that holds events with acquisition of power and status. These event rooms are present in many other dungeon crawler games, such as The Legend of Zelda and The Biding of Isaac, and are very characteristic of the adventure/ dungeon crawler genre. Other kind of events, like mini games or puzzle rooms, can be chosen without much interference to the agent generation method. After choosing the type of the rooms, it is important to design the map positioning of these rooms, which allows the designers to have more control of the order the event rooms appear in the dungeon, and thus, have more control of the level progression. For example, it is interesting that the player finds the equipment events before the boss fight, so he has the feeling of getting stronger at each level and also gets prepared to fight harder bosses. The map positioning will be further discussed in the next subsection.

A. *Map Positioning of Event Rooms*

To begin the map position of the event rooms we considered a square matrix to represent the dungeon space. The dimensions of the initial matrix may vary according to how big it is intended for the dungeon to be. Every cell in the matrix is a position and represents an available room, so that in a 3x3 matrix we have 9 available rooms that can be chosen to form the resulting dungeon.

Following, the first chosen position represents the initial room of the dungeon and this position will be chosen randomly inside the matrix. However, the initial room position is never suited on the matrix border. The reason for this is that by not staying in the border of the matrix, the initial room can have more neighbor rooms and, therefore, originate more paths. The advantage of generating more initial paths is that it results in wider dungeons, since a dungeon with its initial room in the corner can originate only two initial paths, the player initial choice of how to begin the exploration of the level gets too simple.

After this, random positions are chosen in the same amount as of the event rooms and their types are chosen based on the following design rule: the boss room is chosen as the farthest from the initial room, because, as discussed before, it is more interesting if it is reached only after the equipment acquiring rooms are found. No distinction was made between the shop room and the treasure room since they were considered as equivalent equipment acquisition events. In Fig. 3 there is an example of the used room positioning for a dungeon taking place in a 4x4 matrix. For different level constructions, level designers may write and define different rules to be followed by the framework proposed in this paper.

After deciding what type of rooms will be generated at the dungeon and their relative positioning, the next step is the training and implementation of the dungeon generation agents.

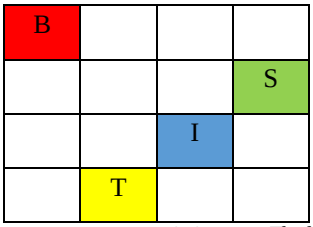

Fig. 3. The event rooms positioning in a 4x4 matrix. The letter I represents the initial room, the letters B, S and T represent the boss room, the shop room and the treasure room respectively.

B. *Dungeon Generation Agent*

We define the Dungeon Generation Agent as the intelligent agent responsible for creating a path to a specific event room and choosing where the event room is going to stay.

In the proposed method, the agents are specialized, meaning that an agent is trained to create the path and define the position of only one event room. To generate the whole dungeon, multiple agents are needed, one for every event room defined in the design phase.

The specialization of the agents brings many benefits to the training stage; one benefit is that the agent actions and observations gets simpler, since they have to decide over only one task, this also allows easier reward mechanics. The other benefit is that the agents can be trained in parallel, and can generate the dungeon parts parallel to each other in runtime,

allowing faster training and faster dungeon generation in the runtime. The parallelization of the agents contributes to debugging as well, since it's clear what part of the dungeon is not being generated according to expected and who is the agent responsible for this part. The agents responsible for finding the shop room and treasure room were not distinguished since in the room positioning stage they were not distinguished too.

The agents were trained using machine learning algorithms available in the Unity Machine Learning Agents Toolkit. This library was chosen due its simplicity of integration with Unity Editor, the game engine where this project was developed. The Unity ML-Agents toolkit is an open source project for training intelligent agents in games or simulation environments. The agents can be trained using implementations of reinforcement learning, imitation learning, neuroevolution algorithms provided by the toolkit, or other machine learning methods through integration with the Python API [11].

The machine learning method chosen to train the dungeon generation agents was the reinforcement learning method provided by the toolkit in its off-policy option. This method implements the Soft Actor-Critic method [11], which is an off-policy actor-critic deep reinforcement learning framework. In this framework, the agents aim to maximize possible reward while maximizing entropy at the same time, representing they try to succeed the task while acting the most random possible. This method has proven to achieve a very nice performance on a range of continuous control benchmark tasks, outperforming prior on-policy and off-policy methods. In addition, this method has proven itself to be very stable, in contrast to other off-policy algorithms, and is able to achieve very similar performance across different random seeds [12]. For its performance capabilities and also the fact that this method is easier to use, since it doesn't require a complex pre-built model or complicated policies to be applied, it was the chosen method to be applied on this project.

The Unity Machine Learning environment has three core entities which are Sensor, Academy and the Agents. The sensors are pre-made components that can be used to observe the agent environment and provide material for the learning. The Academy is used to keep track of the steps of the simulation and manage the agents. The Academy also has the ability to define environment parameters, which can be used to change the configuration of the environment at runtime. The Agents are the main component that indicates which objects are agents in a scene, they are responsible for collecting observations, take actions and receive rewards, and each agent must override these functions to customize the learning experience, and thus, be able to train the agent to the desired objective.

The observations methods are responsible for what an agent is capable of perceiving about the environment, and thus, can considerate in its action taking. The action methods implement the actions an agent can take, like jumping, walking or collecting items. Moreover, the reward methods are responsible for giving rewards or punishment to the agents. In the next subsection, we will discuss in detail the implementation of these methods for the agents in the procedural generation method proposed on this paper.

A square matrix is used to represent the dungeon space in the training environment; every matrix cell represented a possible position for a dungeon room. The actions an agent can take were defined as navigating up, down, to the sides in the matrix and stop creating the path. The agent could take only one move at a time and the place where the agent stopped creating the path was set as the event room that the agent was specialized in.

For the observations, the coordinates of the agent in the matrix were used along with the amount of moves made by the agent. The agent received the biggest positive reward every time it stopped creating the path in the exact position of its designated kind of room or in an adjacent position. Small positive reward was also added after the agent moved to a position that didn't have a room in the matrix. The biggest punishment was added when the agent stepped outside the borders of the matrix and when it stopped its path on a position that was not adjacent to the position of its specialized room nor the specialized room itself.

Each training episode consisted in the agent creating a path of rooms, by marking in the matrix the rooms he had visited. When the agent decided to stop, the room the agent had chosen to stop was evaluated. If the position of the room corresponded to the target position of the event room the agent was specializing in or an adjacent position, then the agent received a positive reward and the episode ended. If the chosen room did not correspond to the target event room or was not nearby, then the agent received a big punishment and the episode ended. At every episode, the position of the event rooms were generated again using the criteria defined in the map positioning of event rooms subsection, which is essential for the training and for the agents to be able to create useful paths in different dungeon dimensions. The training environment was composed solely by the data structures described above, and thus no rendering was required.

The training of the shop/treasure room agent occurred faster than the training of the boss room agent, as we can see in Table 1. This may have occurred because the boss room was always farther from the initial room than the other rooms, so the agent had to make much more moves to arrive at the boss room than at the treasure/shop rooms.

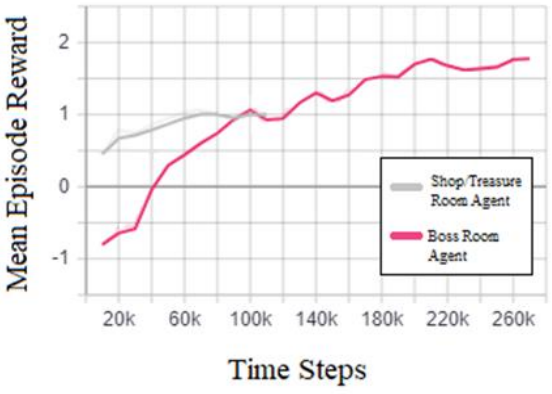

Fig. 4. Mean episodic reward received during training for each agent.

TABLE I. TIME AND MEAN CUMULATIVE REWARD OF AGENT TRAINING

Agent	Mean Cumulative Reward	Training Time
Shop/Treasure Room	1.001	23m 42s
Boss Room	1.772	43m 9s

The total training time for the agents can be considered to be quite satisfactory too. We believe this occurred because the training was structured in a way it was not necessary to render anything on the screen, and thus, not necessary to check for collisions and other costing computing 3D rendering demand, because we were able to instantiate and navigate our agents in simple data structures, such as matrix. As stated in the subsection above, this aspect increases the speeding up implemented in the Unity machine learning agent toolkit used, what makes it possible to run more episodes in less time. This configuration performed better and was easier to be parallelized since lighter environments support multiple agent training at the same time better, which sped up the process. In addition, the final cumulative reward for the boss room was greater than the one for the shop/treasure room, as can be seen in Fig. 4, since the agent was rewarded for every move made to an empty matrix position, and the path length influenced this aspect as well.

The last step is the rendering of the generated dungeon represented in the matrix into a 3D game dungeon. This stage integrate the agents' matrix in a final consolidated matrix, and render the level based on the generated matrix. This can be done by marking every empty room chosen by the agents, represented by the matrix cells, and then copying them to an empty matrix that represents the map to be rendered, using methods to prevent a room marked by an agent to overwrite a room marked by other agent. The 3D rooms that will be rendered in the position of the marked rooms are chosen from an array of pre-made 3D rooms, according to their types, so boss rooms are chosen from an array of pre-made boss rooms, as well as common rooms and all the other types. The doors are positioned at every intersection between two rooms, so between two adjacent rooms there will always be a door.

IV. RESULTS AND DISCUSSION

After analyzing the dungeons generated by the agent generation method proposed by this paper, we could observe that the design choices regarding the positioning of the event rooms were respected, since the boss room is always the farthest when comparing the event room's position with the initial room position, as can be seen in Fig. 5. This conformation forces the player to explore more the dungeon before arriving at the peak of the level progression flow, and thus, delivering a better experience. It is also possible to observe that the shop room and treasure room are often in a path for the boss room, which is positive because it allows the player to equip before fighting the boss of the level, giving a better sense of character development. This configuration can be found in other dungeons crawlers that have its dungeons handmade, such as The Legend of Zelda.

It can also be noticed that the dungeons are not sparse, the pattern where we have direct paths formed by one room following the other until we arrive at the destined room are not likely to occur. This dungeon conformation happens because our agents received a small reward for every empty room they occupied, so, to maximize the reward, the agents ended up also maximizing the number of rooms used to form the dungeon. Because of this aspect, dungeons generated in matrix with small ranges, like 4 or smaller as dimensions of the generation matrix, tend to look a lot alike, since the dungeon is filled with rooms almost to its maximum capacity.

We can also observe that the dungeons follow a characteristic design pattern among them, which is positive since mischaracterization gets far from occurring. This also gives the designer more control of what to expect of the content generated procedurally, being an alternative for the common problem for procedurally generated content, which is lack of control.

The proposed algorithm also showed itself to be very adaptive. The dungeons it generates respect the map positioning of event rooms step, which is responsible for delineating the progression flow of the generated levels, and can be changed according to the game objectives, mechanics or history.

This became possible because of the use of the machine learning agents, which can be trained to a large range of map positionings according to the game designer's needs. In addition, this also contributes to adjust the similarity between dungeons, making it more or less similar according to the designer choices. Therefore, if the game designers are willing to have more diversity in the game, using this generation method, it can be achieved by adding more event rooms or more diverse game rooms positionings and varying them according to the level of the game the player is at.

Fig. 5. Dungeons generated with the agents. A represents a 4x4 dungeon, B represents a 6x6 dungeon and C represents a 9x9 dungeon.

In addition, diversity can be achieved by adding different bosses, enemies and special events held at specific phases, as is done in many dungeon crawler titles. This way, the levels can withstand strong characterization together with diversity in the game.

However, the proposed algorithm demands the agents to be trained with several matrix dimensions for being able to generate dungeons with any dimensions. In the generated dungeons, we could observe that an agent trained in a 5x5 matrix could generate dungeons respecting the design choices up until 10x10 matrix dimensions, for more broader dungeons, it was necessary to retrain the agents in larger matrix so it could still respect the design choices in the generation.

V. EVALUATION

To evaluate the generated dungeon, we conducted a survey with 15 users. In this survey, users were asked to play a simple game where the objective was to find a princess and free her from the cage. In the game, the player had to explore the dungeon searching for the princess and fight enemies that spawned at every common room. The princess was spawned at the boss room, and the shop and treasure rooms were also present. The users were asked to play two versions of this game, one implemented the agent dungeon generation and the other implemented a more manually generated dungeon method where the generation matrix were manually made and were randomly chosen for each level.

The game had infinite agent generated levels, after the player finished a level, the next level was generated adding more one dimension to the generation matrix. So if the player cleared the level generated in a 4x4 matrix, the next one would have a 5x5 matrix as its generation matrix. The levels begin being generated in a 4x4 matrix for both versions. The 4x4 dimension was chosen because the agent generated dungeons were not sparse, so for smaller generation matrix the agent generation method tends to use every possible room, shaping the dungeons as a square.

The survey questions about their experience are shown in Table 2, where version 1 represents the manual version, and version 2 represents the agent version. According to the responses, we can conclude that the dungeons generated by the agents were rated as funnier, with better maps and more replayability values, which is a very positive result since replayability is an important factor for procedurally generated content. The results also showed that there was better user acceptance to the agent generated dungeons.

However, users could guess what dungeon was procedurally generated. This may have occurred because, although they played the same game with only different generated dungeons, the dungeon design is crucial for level balancing in dungeon crawler games. Because of this, the game versions were not equally balanced. This caused the agent generated dungeons to deliver a game experience that was more challenging than the manually generated, which may have lead the users to guess this was the machine learning generated dungeon.

TABLE II. SURVEY QUESTIONS AND RESPONSES

Version 1	***Version 2***	***Indiscriminate***
What game version was funnier?		
26,7%	60%	13,3%
In which game version do you think the maps are better?		
20%	53,3%	26,7%
What game version would play more times?		
26,7%	53,3%	20%
What game version do you believe was generated by an AI?		
6,7%	73,3%	20%

VI. CONCLUSION

This paper presented an alternative for generating dungeons using intelligent agents trained with machine learning, a different approach of the use of agent generation methods found in related works, where the agents were not intelligent and because of it generated poorer dungeons. The proposed method was capable of generating dungeons following the style of iconic games, such as The Legend of Zelda and The Binding of Isaac, but unlike the mentioned games, the proposed method implements machine learning in the generation. This contributed for the adaptability of the method since the use of machine learning in the agent training made it possible to generate different dungeons designs.

The dungeons generated by this work can adapt to different dimensions, being able to create endless levels, but for this behavior, agents need to be trained for different level dimensions, what increases the time of training. Likewise, because of the choice of rewarding the agents for occupying an empty room, the generated dungeons have a tendency to have many rooms and occupy the majority of the available space. The use of pre-made rooms is also a limitation of the method, since it requires a wide number of manually made rooms to add diversity in games with a vast number of levels.

In addition, the generated dungeons have proven to have more replayability and be more enjoyable than manual ones, although users could guess an artificial intelligence (AI) generated them. The incorporation of machine learning in the procedural generation is a powerful tool and Unity ML-Agents is a new and simple resource for game developers to use and improve their content generation capabilities. In addition, the use of machine learning in the procedural generation method improved its versatility and reuse, because it made it possible to adapt the method for various dungeons designs without much effort. For future work, we intend to search possibilities of procedurally placing enemies to balance player progression in dungeons and create robust procedural level generation.

REFERENCES

[1] Mercado de games deve gerar receita de US$ 152 bilhões em 2019. https://epocanegocios.globo.com/Empresa/noticia/2019/06/mercado-de-games-devegerar-receita-de-us-152-bilhoes-em-2019.html. Access in December 2019.

[2] R. Koster, "The cost of games", https://www.raphkoster.com/2018/01/17/the-cost-of-games/. Access in July 2020.

[3] R. van der Linden, R. Lopes, R. Bidarra "Procedural generation of dungeons," in IEEE transactions on computational intelligence and ai in games, vol. 6, no. 1, pp. 78-89, March 2014.

[4] "ML-Agents toolkit overview", https://github.com/Unity-Technologies/ml-agents/blob/master/docs/ML-Agents-Overview.md. Access in August 2020.

[5] N. Brewer, "Computerized dungeons and randomly generated worlds: from rogue to minecraft," in Proceedings of the IEEE , vol. 105, No. 5, pp. 970-977, May 2017.

[6] G. Smith, "An analog history of procedural content generation," in Proc. 10th Int. Conf. Found. Digit. Games, Pacific Grove, CA, USA, 2015, pp. 1–6.

[7] G. Smith, E. Gan, A. Othenin-Girard, and J. Whitehead, "PCG-based game design: Enabling new play experiences through procedural content generation," in Proc. 2nd Int. Workshop Procedural Content Generat. Games, Bordeaux, France. June 2011.

[8] L Johnson, G.N.Yannakakis,andJ.Togelius, "Cellularautomatafor real-time generation of infinite cave levels," in Proc. Workshop Procedural Content Generat. Games, New York, NY, USA, 2010, DOI: 10.1145/1814256.1814266.

[9] N. Shaker, A. Liapis, J. Togelius, R. Lopes, R. Bidarra, "Procedural content generation in games", Chapter 3, Springer International Publishing, 2016.

[10] S. Dahlskog, J. Togelius, "A multi-level level generator" in Proceedings of the IEEE Conference on Computational Intelligence and Games, 2014.

[11] A. Juliani et al. Unity: A General Platform for Intelligent Agents. arXiv preprint arXiv:1809.02627, 2020.

[12] T. Haarnoja et al. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. arXiv preprint. arXiv:1801.01290. 2018.