PROCEDURAL CONTENT GENERATION WITH REINFORCEMENT LEARNING : IDEA TOWARDS PERSONALIZED CONTENT GENERATION

[1]RICHIE BACHTIAR RISMAWAN, [2]WIRAWAN WINARTO, S.T., [3]YUSUF ISLAM S.KOM.

[1,3]Human Computer Interaction Surya University Indonesia
[2]Technical Director Rhema Sistem Abadi Indonesia
E-mail: [1]richie.rismawan14@student.surya.ac.id, [2]winarto@outlook.com, [3]yusuf.islam@surya.ac.id

Abstract- This paper explores the idea of implementing reinforcement learning in procedural content generation (PCG) in video game by letting the generator to observe the state of the environment (player and related object in the level) and generate appropriate content based on policy that it learns. As proof of concept, this idea is implemented into simple endless runner game that generates its obstacles at runtime. The result of the experiment shows that the agent is able to learn and generate more fitting content than using random-number based generator. However, the result reported in this paper are based on simplified environment. As such, the paper concludes that there are still more potential that can be explored in this idea which could contribute into a more personalized content generation. In addition of extending the idea further than video game context.

Keywords- Procedural content generation, reinforcement learning, personalized content generation, video games development

I. INTRODUCTION

In Video Game development, especially those that falls into "Roguelike" genre, using a method called Procedural Content Generation (PCG), is one of the most common method to make a game with high replay value. Procedural Content Generation (PCG) itself, as defined by Shaker, Togelius, and Nelson in their book, is an algorithmic approach of creating game content with limited or indirect user interaction [1]. The term of game content itself in general are used to represent some of the component within the game. Those components could be categorized as, Game Bits (most basic component in game like textures and sounds), Game Space (area or level of the game), Game System (rules, or behavior), Game Scenarios (events or sequence), and Game design (the whole goal and rules) [5]. For example, one could make a game with random generator which function is to spawn objects across the level, by doing so, every experience that player receive will be different and should increase its replay value. This method could also help game developer and designer because they could make countless levels and pattern with less effort than trying to design each level manually.

However, because most common PCG methods are usually using a random seed to generate each level (usually stochastically), game developer or designer tends to have less control over the environment most of the time. Thus, often we found players getting overwhelmed by the difficulty. Even though we as player or developer could say that it was the essence or challenge of procedurally generated games, in the end of the day, we still need to thrive to develop the best experience for the player, which could not be achieved if the players being overwhelmed repeatedly. As such, a more personalized approach is needed in PCG [1].

The vision of more personalized PCG have been discussed by Yannakakis and Togelius in their paper as "Experience-driven procedural content generation (EDPCG)" which is a framework for PCG built around the idea of generating personalized content experience based on computational user modelling [2]. EDPCG itself consists of four main components, (1) player experience modelling, (2) content quality, (3) content representation, and (4) content generation. Each of these components have their own function in the framework. Mainly, they function to, (1) create player-game experience model, (2) assessing generated content in regards of player experience, (3) decide on how the content itself will be represented, and (4) the action of generating the content itself.
Another similar research in this context had also been done in more practical sense. Shaker, Yannakakis, and Togelius have done research on implementing PCG with preference learning based on player model which could predict player experience based on features of level design and playing styles [3].

In modelling the player experience itself, Pedersen, Togelius, and Yannakakis have tried to build a player experience model using "Super Mario Bros" game as its base. This research mainly focuses on the relationship between parameters in levels, playing styles, and user experience to create a neural network model which could predict player experience in the form of challenge, fun, and frustration. The result of this research shows that a single-layered perceptron could predict challenge and frustration with fairly high accuracy (77.77% for challenge, and 88.66% for frustration). Although, the model could only predict

fun with 69.18% accuracy which shows that there may be a need for more non-linear classifier [4].

As we can see, there are many research that lean towards player or user centered PCG with the help of machine learning especially in context of player modeling. However, most of the research that has been done could only predict or adjust itself if there are training data with an input-output pair. In other words, the system need to know the "facts" or correct answer before it can learn. This could be problematic because most of the time, game developer cannot subjectively judge or know whether the output itself is true. In addition, these learning methods mostly used only to classify each player, but not have a direct impact on generating the content itself. Classifying player and passing certain parameter to the content generator by itself, is not different than using an "if" statement within the system.

Thus, this paper will try to introduce new idea of PCG model by implementing machine learning method called reinforcement learning, which can learn a policy based on observing its environment and optimizing reward. In hopes that the result of this research will lead towards a more personalized PCG by learning the environment (and player indirectly) without discrete input-output pair. As such, this paper will be organized as follows. Section 2 will give a brief explanation about reinforcement learning in general. Section 3 will introduce the system design and potential challenges. Section 4 shows experimental implementation and its results. Lastly, section 5 will conclude the paper and present the possibilities of the research in the future.

II. REINFORCEMENT LEARNING

Reinforcement learning is one of machine learning method with a core concept of learning a policy based on observations of current states and exploring consequences of actions sequence. Reinforcement learning by itself are commonly used to build or train an autonomous agent that can sense its environment and choose the most optimal actions. In other words, reinforcement learning is learning through mapping observation states and sequence of actions, with a goal of finding the most rewarding actions sequence for each situation [6]. Similar to other learning method such as supervised learning or unsupervised learning, reinforcement learning also do attribute selection and model selection. In case of reinforcement learning, the attributes are the set of observation (called observation vectors) which used to help the agent know information that could help on solving the problem. Meanwhile, the model selection itself is creating a policy (mapping observation states to actions) [7].
A reinforcement learning system could be identified by these elements, (1) a policy that defines agent's behavior at given time, (2) a reward signal that defines the goal in learning problem by giving rewards as agent tries to maximize it in more immediate sense, (3) a value function which defines what is good for the agent in the long run, and optionally (4) a model of the environment which makes plan for the agent or predict future rewards and states before actually doing it. [8]

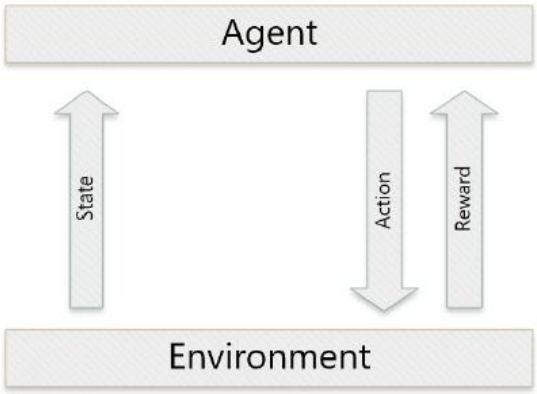


Fig. 1. Reinforcement Learning Cycle

Define,
$S = States\ of\ environment$
$A = Set\ of\ Agent's\ Actions$
$t = Discrete\ time\ step$
$r = Reward$
$\delta = Successor\ State$
$\pi = Policy$
$V^{\pi} - Cumulative\ value = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$, with $0 \leq \gamma < 1$

Goal: Learn $\pi : S \rightarrow A$ where $V^{\pi}(s) \rightarrow Max\ for\ each\ s$

- For each time step (t), observe current states (s_t)
- Choose action (a_t) and perform it
- Receive reward from environment $r_t = r(s_t, a_t)$
- As an effect of action, new state $s_{t+1} = \delta(s_t, a_t)$ is created
- Do evaluation and learn policy that maximize V^{π}

Fig. 2. Reinforcement Learning Process

Most of the reinforcement learning algorithms are used to addressing a problem in a setting known as Markov decision process (MDP). More specifically, in MDP which the transition probabilities are not explicitly defined. However, there are several things to note on reinforcement learning as follows [6].

1. **Delayed reward**: Because reinforcement learning only gives reward as the agent do actions, the agent need to determine how much it should care about future rewards. This is important because the agent's actions also determine the new state of the environment and the next action it would take. Poorly designed reward could make the agent ends up in a less significant reward or even fail at its tasks. So, the agent needs to learn which actions that could make a better eventual reward value, not just its current immediate reward value.
2. **Exploration and Exploitation:** Determining an exploration strategy is also important in

reinforcement learning. Exploration strategy could make learning process more effective or the opposite. Choosing tradeoff between exploring new states and exploiting known ones is the key to create an effective learning process.

3. **Partially observable states:** In some situations, some sensors could only detect partial information. This makes it important to consider working together with another agent on collecting information from the environment.

The advantages reinforcement learning itself are its capabilities to learn by using its own experience. Unlike supervised learning where training is done by using labeled data from external supervisor with clear specification of correct action the agent should take in a situation, this kind of learning is more intended for generalize its response for new situation that is not present in training data. Reinforcement learning also differ from unsupervised learning in terms of its purpose. Even though reinforcement learning and unsupervised learning both does not need a clear correct action, unsupervised learning is leaned towards finding hidden structures in data which does not really answers the problem of maximization of rewards that reinforcement learning does [8].

III. SYSTEM DESIGN AND CHALLENGES

By concept, the system proposed in this paper are using reinforcement learning to learn a policy from the environment which player continuously interact with. By observing the environment which player alters, it hopes that it can indirectly learn the player and generate content based on player behavior. The component of the system itself are divided into four main components as follows.

1. **Environment:** Consists of every state in the environment within the game in respect to its importance and role for deciding sequence of action for content generation. Every parameter on player (health, position, time played, etc.) or the condition in the level (enemies left, time left, distance between player and goal, etc.) could be used as state to be observed.
2. **PCG-Agent:** An AI agent that responsible for generating content for the environment (level). The decision that this agent took are derived from Agent Brain. One environment can contain one or more agent each with same or different function. In other words, PCG-Agent could be described as an actuator which do sequence of action instructed by its brain.
3. **Agent Brain:** This component is mainly used for making decisions and updating the neural network model (training) using reinforcement learning. Every decision made are done in observe-decide-action-reward cycle. In other words, this component is responsible for finding the most optimized sequence of action based on environment states that could give the most reward for the agent.
4. **Player:** The entity that interact with the environment, changing its states. This component is used as base for the system to decide what and how the content should be created. All with a purpose to create the most fitting experience for the player itself.

The flow of the system itself on each session can be described as follows.

- On game environment state change in gameplay and PCG-Agent is not marked as done:
  - PCG-Agent observe the new states and pass its parameter (observation vector) to Agent Brain (request new decision).
  - Based on passed parameters, action sequence instructed for each agent.
  - On each agent with new instruction, generate content based on the instruction. This changes states on the environment.
- On immediate reward, add its value to cumulative reward.
- On condition where the game needs to be stopped (clearing the level, the player dies, time limit, etc.) mark PCG-Agent as done.
- On done, add any delayed reward to cumulative reward.
- Give reward feedback to Agent Brain.
- Based on given reward, reinforce the policy (do optimization).

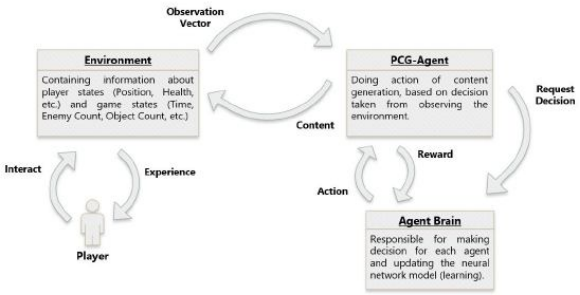


Fig. 3. Relation between each component in the proposed system

These steps show how reinforcement learning will be used in our design. Because reinforcement learning need to learn policy by mapping sequence of action and rewards, in this design we put the action as content generation and rewards in two types, as immediate effect of an action and as result of sequence of actions. In doing so, every content generated on runtime will be based on player and current environment observations, with addition of keep reinforcing the policy in every play session which in turn could make a more appropriate content generation.

However, because of the nature of reinforcement learning itself, we need to consider some potential challenges on implementing this design.

1. Reward function

This focuses on the problem of when and how the reward will be given for the agent brain. One could

use a reward function which gives a reward when the agent fulfills the goal. Otherwise, we could give nothing or negative reward when the agent fails. Another could make a negative reward for each time step, so that the agent brain must learn sequence of actions that is most effective and can be done in short time period. This is a main component that must be addressed, especially in context of procedural content generation because of the fact there is no actual correct way to create the content. In considering this, the designer or developer need to answer some question such as, should the brain get reward when it generates content, should it wait until the end of play session, or should it get both immediate reward and delayed reward. In addition, we should determine how much reward that the brain gets, in respect to how the content is created. One could also try to give the player a chance to reward the generator itself in a form of feedback, although this form of reward could also open the possibilities for the player to give false reward and manipulate the system, which in some cases are not desirable from the developer or designer point of view. Designing reward so it could represent "correct" action in this context, is the key on creating a good learning environment. The process of designing reward are often trial and error. As such, designer and developer should try and look at agent behavior during training and predict whether the brain could produce desirable results.

2. Length of training time

Like other machine learning methods, reinforcement learning still need time to do training before it become reliable. As such, one potential challenge that this system encounter is how long should the agent learn the player and how to speed up the training process. Because every agent brain in the system that this paper proposes is constantly trained in runtime (play session), we need to specify when the agent should stop giving rewards and instead just request decision from brain. This way, the brain could just use the model that it already created to make decisions. Extensive training could lead the agent to become too specific and mitigates the purpose of PCG itself. Also, we should note that in complex environment, the training process could take a long time and makes the system impractical to implement. One way to speed up the training process is by connecting the brain with multiple agent. This could speed up the training process because each agent could experience different observation-action-reward cycle.

3. Content variance

This is related to the trade-off between exploitation and exploration in reinforcement learning. As we know, reinforcement learning method are learning by producing sequence of actions until it found a policy that gives the most reward in every state. This could lead to monotonous behavior from the agent which exploit its known actions, or a stochastic behavior from the agent that always explores new sequence of actions. Although it is true that the purpose of this system is to create a more personalized PCG, the system itself still need to avoid generating the same content repeatedly just because it gives the most reward. In implementing the system, the designer or developer need to consider the possibilities of players trying to exploit the algorithm. One way to prevent this is by limiting the frequency of retraining the agent brain, by only train the agent brain if there are a set amount of difference in player actions or behavior. Another way to prevent this, is by shortening the training phase so that the agent brain will still maintain a bit of randomness. Finding the right balance could prove to be difficult and need some process of trial and error.

IV. EXPERIMENTAL IMPLEMENTATION

A. Environment Settings

This experimental implementation is done within Unity3D game engine and utilizing its machine learning agent library which is based on tensor flow open source library [9]. The machine learning agent library itself are using reinforcement learning technique called Proximal Policy Optimization (PPO) which use neural network to approximate ideal function that able to map observed states to agent action [10].

The system itself are implemented into a simple endless runner game which the PCG-agent are tasked to procedurally generate obstacles at runtime for the player to avoid. In this game, the generator need to adjust itself at runtime, so it could generate sequence of obstacles that is challenging for the player. Rules of the game are as follows.

1. Player are given 3 starting health.
2. When player collides with an obstacle, subtract the player health by 1.
3. If the player has 0 health left, the player loses.
4. If the player could survive for 30 seconds, the player wins.
5. The level consists of 5 different lanes where player and obstacles are placed.
6. The player can move vertically to adjacent lanes.

In this context, the content is classified as "challenging" if the obstacle is spawned within the move range of the player. In example, if the player is in lane 3, that lane and the lanes that is adjacent to it (lane 2 and 4) are considered as in the move range of the player.

In that respect, the PCG-Agent for this game are designed to observe current player position and number of obstacle on each lane. Those features are chosen because they are considered as main aspect of generating appropriate obstacle placement. In generating the content, the agent is given a set of actions that is, (1) do nothing and (2) spawn obstacle on lane 1-5.

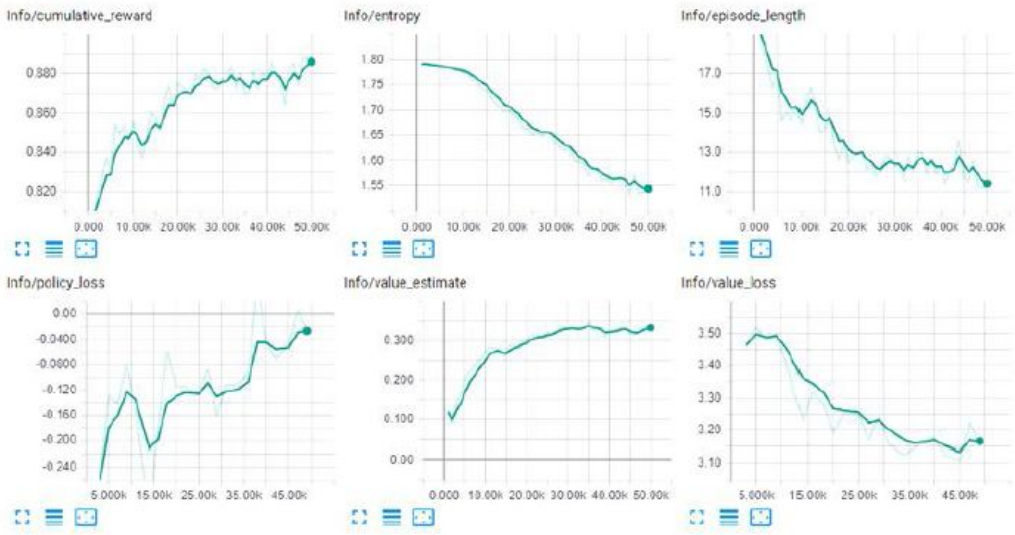


Fig. 4. Agent training summary graph (taken using tensorboard)

The reward signal itself are designed to the following.

- -0.01 rewards at every step
- +1 rewards when player dies
- +0.5 rewards when player survive with 1 heath
- +0.25 rewards when player survive with 2 or more health

These design choices are taken under the consideration of information, action, and reward signal that the agent needed to be able to generate the content effectively.

B. Training Phase

In this experiment, the training is done using PPO with some training hyperparameters are set to be as follows. Note that these hyperparameter are chosen based on trial and error when implementing the system, hence it should be different for each game.

Table I. Hyperparameter settings

Hyperparameter	Value	Description
gamma:	0.8	Value that correspond whether the agent should care for possible future rewards. Larger number indicates that it should care more.
batch_size:	128	How many experience (observe-decision-action-reward) cycle for one gradient descent update iteration in learning process.
buffer_size:	2048	How many experience should be collected before the agent do the learning process (updating the model).
learning_rate:	1.0e-3	Corresponds to the strength of gradient descent update step.
time_horizon:	128	How many steps of experience to collect before passing it to experience buffer. The value need to be adjusted to capture all important behavior
num_layers:	1	Hidden layers that present after observation input.
hidden_units:	128	Units for each fully connected layer of the neural network.
beta:	5.0e-3	The strength of entropy regularization. Increasing this value will make the agent take more random actions in order to explore the action space.
max_steps:	5.0e4	Maximum time step in simulation

Because of limited time constraint, using real player for training in this experiment are considered to be impractical. Hence, the player in this experiment are simulated using algorithm that moves the player using probability of 40% to move up, 40% to move down, and 20% to do nothing. The player itself are designed to take actions every 0.5 seconds. The play session (training episode) will reset automatically whether the game reach win or lose condition. This training will continue until it reaches its defined maximum step. The training itself will be done within 50000 time step, with 1 time step correspond to 1 frame in the game. As such, if the simulation is maintained to be running at 60 fps (frame/second), then the simulation will be done in approximately 14 minutes.

In addition, because the player itself are simulated, the experiment itself can be designed to have a faster global time scale. In this experiment, the training is done with five times more speed than normal. Hence, if the simulation could run at 60 fps, then with 5 times speed, the simulation could run at 300 fps. This is significantly useful in this experiment because learning process could take a long time before it produces results. This also save time and resources than using real people, even though it must be noted that using real people will give more accurate results.

C. Training Result

Performance of the agent are monitored using tensorboard [11]. As we can see from the graph (Fig.4) we can get information as follows.

1. The rewards received to the brain are increasing over time. This means that the agent is able to learn and getting more reward over time.
2. The value in value estimate (predicting its future reward on each state) are increasing as the value in value loss decreasing. This means that the agent could estimate the value of each

state well and able to predict future reward that it will get at any given point.

3. The entropy (randomness of choice) of the agent itself are slowly decreasing. This means, that each decision the agent made are starting to be determined by policy that it learn.
4. The length of each training episodes is shortened from time to time. This means, that the agent is reaching its goal faster over time.
5. The magnitude of policy loss (how much the policy changes overtime) are fluctuating at first but slowly decreases when the training stabilizes. This means that the policy that it learned are starting to stabilize.

Based on that information, we could prove that the agent is successfully trained to observe the player and generate content that could defeat the player (which in this case proves that the agent is able to create challenging content for the player).

As shown during runtime, using the trained agent (Fig. 5.) will generate more fitting content generation (in this case, obstacle placement) than using untrained agent or random number generator (Fig. 6.). The agent is shown to generate the obstacle in a way that have higher probability to be hit by player. Even so, when the player is in the middle, the agent is shown to generate content more randomly. This could be caused by the player algorithm that never stay exactly in the middle. However, it should be noted that this experiment is done in a simplified environment. Implementing this design in a more complex environment could be difficult, especially on designing the reward and adjusting the training parameters.

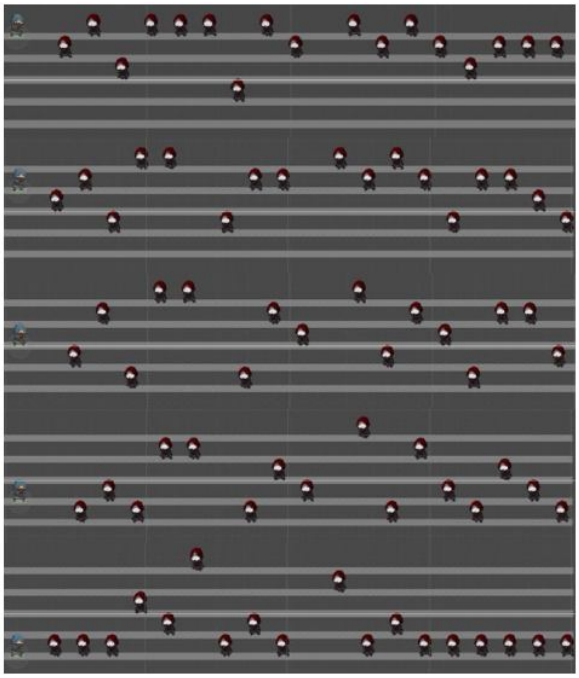


Fig. 5. Generated level using trained agent based on current player position *(from top-bottom)*

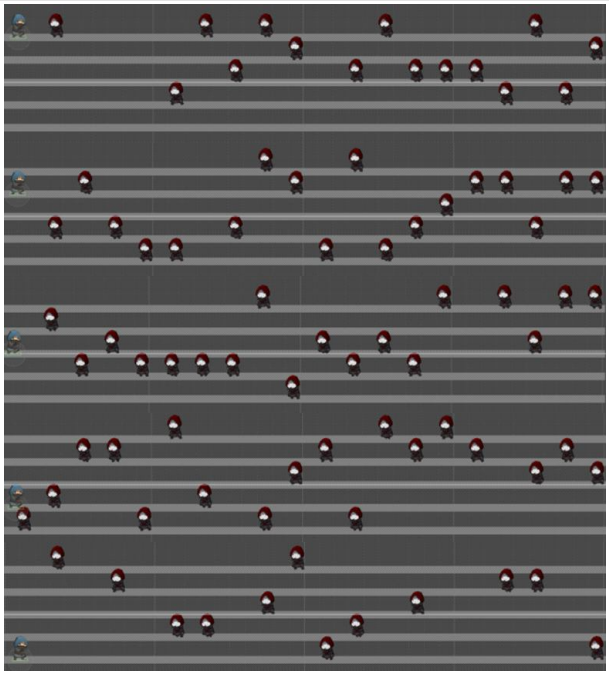


Fig. 6. Generated level based on random number generator *(from top-bottom)*

CONCLUSION AND FUTURE WORKS

This paper has explored an idea and a proof of concept on utilizing reinforcement learning method in context of procedural content generation. Although the PCG agent reported in this paper able to generate fitting content, it may be difficult in a more complex environment. As such, there are still many room of possibilities to expand and refine this idea. Additionally, this idea could also be generalized for context other than video games.

REFERENCES

[1] Shaker, N., Togelius, J., & Nelson, M. J. (2016). Chapter 1 - Introduction. In N. Shaker, J. Togelius, & M. J. Nelson, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research* (pp. 1-15). Springer.

[2] Yannakakis, G. N., & Togelius, J. (2015). Experience-driven procedural content generation. Affective Computing and Intelligent Interaction (ACII), 2015 International Conference on 2015 Sep 21, 519-525.

[3] Shaker, N., Yannakakis, G. N., & Togelius, J. (2010). Towards Automatic Personalized Content Generation for Platform Games. AIIDE.

[4] Pedersen, C., Togelius, J., & Yannakakis, G. N. (2009). Modeling player experience in super mario bros. Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on 2009 Sep 7, 132-139.

[5] Hendrikx, M., Meijer, S., Van Der Velden, J., & Iosup, A. (2013). Procedural content generation for games: A survey. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), 1.

[6] Mitchell, T. M. (1997). Chapter 13 - Reinforcement Learning. In T. M. Mitchell, *Machine Learning* (pp. 367-390). McGraw-Hill Science/Engineering/Math.

[7] Unity Technologies. (2018, March 24). Background: Machine Learning. Retrieved from https://github.com/Unity-Technologies/ml-

agents/blob/master/docs/Background-Machine-Learning.md

[8] Sutton, R. S., & Barto, A. G. (1998). Chapter 1 - The Reinforcement Learning Problem. In R. S. Sutton, & A. G. Barto, *Reinforcement Learning : An Introduction* (pp. 1-25). MIT Press.

[9] Unity Technologies. (2018, April 5). ML-Agents-Overview. Retrieved from https://github.com/Unity-Technologies/ml-agents/blob/master/docs/ML-Agents-Overview.md

[10] Unity Technologies. (2018, March 15). Training with Proximal Policy Optimization. Retrieved from https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-PPO.md

[11] Unity Technologies. (2018, March 12). Using TensorBoard to Observe Training. Retrieved from https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Using-Tensorboard.md.

★★★