Procedural Content Generation for Tower Defense Games: a Preliminary Experiment with Reinforcement Learning

Yueming Xu[1,a)] Tetsuro Tanaka[2,b)]

Abstract: Although procedural Content Generation via Machine Learning (PCGML) has recently enjoyed considerable popularity, there is little research on PCGML applied to more complicated games such as Tower Defense (TD). We trained agents to play TD levels (the solver) and generate TD levels (the generator) using reinforcement learning on a TD simulator developed in-house. We conducted the experiments of solver agents with different action spaces to find the most proper one for our task. Then we tried to generate levels, but the results showed that there is still a lot of room for improvement.

Keywords: Procedural Content Generation, Reinforcement Learning, Tower Defense

1. Introduction

Procedural Content Generation (PCG) is the process of algorithmically creating game assets. PCG can be employed to increase games' replay value and reduce the production cost and effort for the games industry[5]. Some forms of game content such as trees and landscapes have been generated procedurally for a long time. However, many traditional PCG usually needs developers to do much hand-coding work.

Machine learning has achieved great success in content production, including generating audio, photo, and other content types across different domains. It stands to reason that machine learning would be applicable to games content generation. One relatively new paradigm of PCG called Procedural Content Generation via Machine Learning (PCGML)[15] has had enjoyed considerable popularity recently.

In this research, we will focus on level generation based on reinforcement learning. As the level design is costly, there are usually too limited game levels available to generate playable levels through data-driven machine learning methods. To address this challenge, for one thing, researchers proposed variants of supervised learning, such as TOAD-GAN[4] and CESAGAN[16], which need only one example or a few examples to generate levels similar to sample levels. For another thing, which is our core focus, reinforcement learning-based PCG[11] methods have begun to be studied for their role in yielding various and playable levels without sample data recently [8].

The tile-based dungeon is one of the most popular testing grounds for level generation, in which levels are made up of different types of tiles, including walls, roads, collectible items and even enemies. Many previous works [16], [11] have made outstanding contributions to dungeon generation. However, there is little research on PCGML applied to more complicated games such as Tower Defense games.

Tower Defense (TD) is a popular casual game genre that has proven to be a good testbed for AI and game research.[3]. At the present stage, we mainly focus on approaches based on adversarial reinforcement learning (ARLPCG)[8], in which the solver agent and generator agent perform generate-and-test iterations to create playable levels. These two agents should be trained iteratively, i.e., when one network's parameters are updating, the parameters of the other network are frozen. Nevertheless, the number of previous works using reinforcement learning to play TD games or generate TD games' levels is limited, so it is necessary to do the preliminary experiments of solvers and generators separately to examine whether reinforcement learning agents can compute proper policy for TD games. Besides, due to the lack of high-quality open data for modern TD games, we developed a TD game in-house based on the rule of a popular mobile game called *Arknights*[1] as our test subject.

2. Background

2.1 Tower Defense Game

The most common form of tower defense game is a single-player game in which the player aims to strategically place defensive towers on the game's map to fight waves of invading attackers. The player will be granted a set amount of gold to begin with, which may be spent to place and improve towers. By killing enemies, the player may gain additional gold. The player's life count will be reduced if enemies arrive at the map's exit points. When all enemies have been defeated or the player's life count lowered to zero, the level ends.

2.2 The rules of Arknights

Our testbed is modeled on *Arknights*, which has tile-based maps, see Figure 1. There are the following main differences between common TD games and *Arknights*:

- Deployment Points (DP), functionally identical as golds in

[1] Graduate School of Arts and Sciences, The University of Tokyo
[2] Information Technology Center, The University of Tokyo
[a)] xu-yueming863@g.ecc.u-tokyo.ac.jp
[b)] ktanaka@g.ecc.u-tokyo.ac.jp

Fig. 1 A screenshot of *Arknights*

other TD games, regenerates at the rate of 1 DP per second. Furthermore, unlike standard TD games, players will get DP when the specific towers instead of all towers kill enemies.

- Instead of upgrading towers when challenging levels, players can upgrade towers before games' levels start, like role-playing games. Therefore, there is a recommended average level of towers for each level in the original *Arknights* game.
- Towers will stop enemies' moves if they are placed on the enemies' path. Melee enemies will attack the tower that is blocking it, and ranger enemies will attack the tower in the attack range.

3. Related Research

Reinforcement learning (RL) is one of three fundamental machine learning paradigms, alongside supervised learning and unsupervised learning, and is generally modeled as a Markov Decision Process (MDP). The MDP describes the transition of an agent between different states, and transition probabilities between states are determined only by the current state and the agent's actions. After each state transition, the agent is given a reward. The RL agent aims to maximize the expectation of the cumulative discounted rewards.

Andersen et al. [2] introduced a real-time strategy game environment with a relatively more uncomplicated observation space, which attempts to fill the gap between Atari 2600 and Starcraft II games. To examine whether the game environment is working properly, they used standard DQN from Mnih et al.[13], a variant of DQN, rule-based, and random policy to play the game. However, as a result, they found that the performance of learning-based methods can not advantage over the rule-based methods.

Liu et al. [12] presented a framework based on evolutionary search, which is capable of automatically generating levels with a similar style of existing content for TD games. A reinforcement learning agent was also used to determine a winning policy and a numerical difficulty evaluation for each created level. Their experimentation created three levels, later tested by humans and considered to be enjoyable.

Khalifa et al. [11] proposed a framework for 2D level generation using reinforcement learning. PCGRL framework models the generation process of levels as an MDP. In this process, the generator iteratively modifies the level toward given goals. Earle et al. [6] presented an approach that is capable of producing controllably diverse levels based on the PCGRL framework by using conditional input and reward shaping.

Gisslén et al. [8] introduced a model based on adversarial reinforcement learning, which can improve the generalization of a reinforcement learning agent and generate levels for 3D games.

4. Proposed Methods

In this research, we developed a clone of *Arknights* as our tower defense simulator. Then we trained the solver and generator agents through reinforcement learning with PPO[14]. We will introduce our tower defense simulator, solver, and generator agent details in the next subsections.

4.1 The Tower Defense Simulator

As mentioned above, *Arknights* is a TD game with role-playing games' elements necessitating more intricate level design and difficulty balance in order to provide a satisfying player experience. That is why we take *Arknights* as our tower defense simulator's basic rule. We developed the tower defense simulator with Unity [*1] based on wiki[*2][*3] written by players (see Figure 2).

4.2 The Solver

We trained the solver through reinforcement learning with PPO[14]. Generally, we defined map size observations and vector observations. We adjusted the IMPALA ResNet's implement [7] by removing the max-pooling layers and used it to process the map size observations. Figure 3 shows how the game-states are represented as map size observations. The detail of the map size observations is presented in Table 1. The vector observations will be concatenated to flatten map size observations before feeding to full-connected layers. The detail of vector observations is presented in Table 2.

Table 1 The map size observations

Observations	Type	Numbers of channels
The type of tiles	one-hot	8
If in towers' attack range	one-hot	36
Number of flying enemies	one-hot	6
Number of normal enemies	one-hot	6
Number of ranger enemies	one-hot	6
Number of melee enemies	one-hot	6
Number of enemies who can not attack	one-hot	6
Number of enemies in the next wave	one-hot	6
Physical DPS of existing enemies	float	1
Magical DPS of existing enemies	float	1
Defense of existing enemies	float	1
Magical resistance of existing enemies	float	1
Current HP of existing enemies	float	1
Number of enemies towers can block	one-hot	6
Skill's state of towers	one-hot	5
Defense of existing towers	float	1
Magical resistance of existing towers	float	1
Current HP of existing towers	float	1
Melee DPS of existing towers	float	1
Ranged physical DPS of existing towers	float	1
Ranged magical DPS of existing towers	float	1
HPS of existing towers	float	1

Generally, the solver agents can place towers, destroy existing

[*1] https://unity.com
[*2] http://prts.wiki/
[*3] https://map.ark-nights.com/

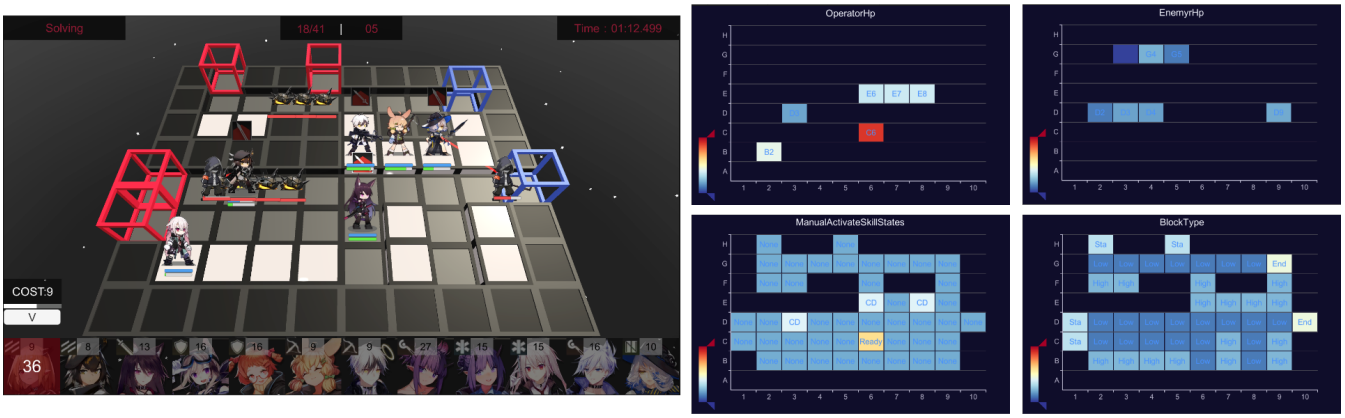


Fig. 2 A screenshot of the tower defense simulator based on *Arknights* with some characters as towers (left). Some representations of the game state (right).

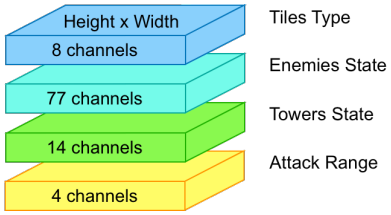


Fig. 3 Illustration of how the observations of game-state are represented as matrices

Table 2 The vector observations

Observations	Type	Length
Current DP	integer	1
Player's current HP	integer	1
Towers' DP costs	integer	12
Towers' type (ranger or melee)	one-hot	24
Towers' type (attacker or healer)	one-hot	24
Does player have enough DP to place towers	one-hot	24

towers, and activating existing towers' skills (e.g., generate additional DP). Solver agents receive a small positive reward when towers attack enemies and a significant positive reward when they kill an enemy. They get a negative reward for failing to prevent enemies from reaching players' territories. Besides, we also add some auxiliary rewards. The detail of the rewards is presented in Table 3.

To find the proper action space for the solver, we trained the three kinds of solvers having different sizes of action space, which we named *Wide*, *Medium*, and *Narrow*.

Table 3 Reward of the Solver of the Solver

Action or Event	Reward
Take an action	-0.5
Take an invalid action	-1
Tower is destroyed by player in 3 seconds after placing	-0.5
Tower heals another tower	0.05
Tower hurts enemies	0.05
Tower kills a enemy	1
Tower is destroyed by enemies	-5
Player's lives count is deducted	-10

4.2.1 Wide

This solver has the largest action space, including valid and invalid actions. The valid action is what can be executed according to current environment states. In contrast, invalid action means that it will not be executed even though the solver chooses to do (e.g., try to activate a tower's skill that is not ready). The solver agent will get a negative reward when it tries to do an invalid action.

4.2.2 Narrow

Opposite to *Wide*, this solver is allowed to execute valid actions only.

4.2.3 Medium

This solver is allowed to do a part of invalid actions. In current experiments, we only forbade the solver from taking the action of placing a tower without enough DP.

Table 4 Types of tiles

Tile	Tile's name	Description
	Invalid	padding of map
	Low	component tile of ground path
	High	component tile of high platform
	Low Forbidden	towers can not be placed on
	High Forbidden	towers can not be placed on
	Start	enemies' spawn point
	Drone Start	flying enemies' spawn point
	End	players' territory

4.3 The Generator

As mentioned above, in this part, we will only discuss how we train the map generator individually (i.e., the generator and the solver are not trained iteratively).

Similar to the solver, we trained the map generator through reinforcement learning with PPO. However, we used a smaller CNN implementation proposed by Gudmundsson et al. [9] to process the observations because the observation space of the generator is tiles types of the map, which is relatively small. The generator can change the type of any tiles in the current map (corresponding to *Wide Representation* in [11]). Before training, we set a series of goals for the generator, such as suitable numbers of specific kinds of tiles. When the agent does the action making current states closer to the goal, it will receive a positive reward,

and if farther, it will receive a negative reward. In current experiments, we use eight different types of tiles, see Table 4, and maps will be randomly initialized when each episode begins.

5. Experiments

In this section, we will introduce the preconditions of our experiments and the solver and generator results.

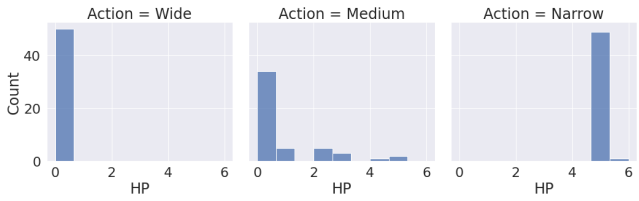


Fig. 4 Results of the solvers

5.1 Preconditions

We conducted the experiments of training the solver and the generator for generating maps individually by using the Unity ML-Agents Toolkit[10]. In this toolkit, neural networks are implemented using PyTorch. Our training programs are run on Google Colab with Tesla K80 GPU. The detail of the environment is presented in Table 5.

Table 5 The detail of the environment

Name	Version
Python interpreter	3.7.12
PyTorch	1.7.1
CUDA	11.0
ml-agents	0.26.0
Unity	2021.1.1.12f1

5.2 The Solver

We trained three solvers for 100000 steps in a level consisting of the map (see Figure 6(c)) and the certain waves of enemies, which can not be cleared by a random player but can be cleared by experienced human players. The learning curves are presented in Figure 5. Then, we let solvers make inferences for 50 episodes in the same environment where they are trained, and Figure 4 shows the results. The players will get 10 HP when the game begins, and if an enemy reaches the player's territories, players will lose 1 HP, i.e., 0 HP means the solver could not clear the level.

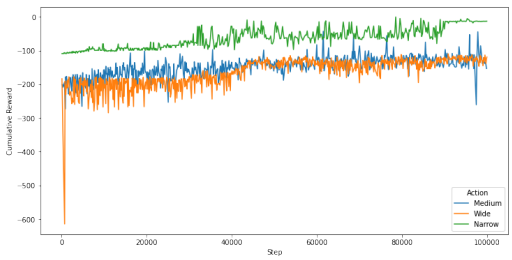
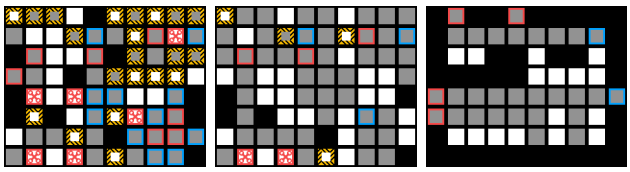


Fig. 5 The learning curves of agents

5.3 The Generator

Figure 6 shows the results after training 1000000 steps. We can find that the generated level is more natural than the initial level. However, compared to Figure 6(c), there are no obvious path patterns in our results.



(a) Initial (b) Generated (c) Handcrafted

Fig. 6 Result of generator

6. Conclusions and Future Work

A conclusion we can draw from the results of the experiment is that we can train an agent which is capable of clearing levels of *Arknights* with PPO and ResNet. Besides, we find that the solver agent with *Narrow* action space tends to have the highest success probability. We can find that the generated level is more natural than the initial level. However, compared to Figure 6(c), there are no obvious path patterns in our results. We need to make more efforts to improve the performance of the generators.

The most significant future work is to apply adversarial reinforcement learning based methods to TD games. To achieve this, we need to tune the model of the generator or carry out experiments using different action spaces and observation spaces for generators since the result is undesirable. Then we should find appropriate ways to analyze the performance of solvers and generators.

This work was supported by JSPS KAKENHI Grant Number JP18K11600.

References

[1] Arknights. https://www.arknights.global/. (Accessed on 07/07/2021).
[2] P.-A. Andersen, M. Goodwin, and O.-C. Granmo. Towards a deep reinforcement learning approach for tower line wars. In *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 101–114. Springer, 2017.
[3] P. Avery, J. Togelius, E. Alistar, and R. P. Van Leeuwen. Computational intelligence and tower defence games. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 1084–1091. IEEE, 2011.
[4] M. Awiszus, F. Schubert, and B. Rosenhahn. Toad-gan: Coherent style level generation from a single example, 2020.
[5] A. M. Connor, T. J. Greig, and J. Kruse. Evaluating the impact of procedurally generated content on game immersion. *The Computer Games Journal*, 6(4):209–225, 2017.
[6] S. Earle, M. Edwards, A. Khalifa, P. Bontrager, and J. Togelius. Learning controllable content generators. *arXiv preprint arXiv:2105.02993*, 2021.
[7] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR, 2018.
[8] L. Gisslén, A. Eakins, C. Gordillo, J. Bergdahl, and K. Tollmar. Adversarial reinforcement learning for procedural content generation, 2021.
[9] S. F. Gudmundsson, P. Eisen, E. Poromaa, A. Nodet, S. Purmonen, B. Kozakowski, R. Meurling, and L. Cao. Human-like playtesting with deep learning. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2018.
[10] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange. Unity: A general platform for intelligent agents, 2020.

[11] A. Khalifa, P. Bontrager, S. Earle, and J. Togelius. Pcgrl: Procedural content generation via reinforcement learning, 2020.

[12] S. Liu, L. Chaoran, L. Yue, M. Heng, H. Xiao, S. Yiming, W. Licong, C. Ze, G. Xianghao, L. Hengtong, D. Yu, and T. Qinting. Automatic generation of tower defense levels using pcg. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, FDG '19, New York, NY, USA, 2019. Association for Computing Machinery.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017.

[15] 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), 2018.

[16] R. R. Torrado, A. Khalifa, M. C. Green, N. Justesen, S. Risi, and J. Togelius. Bootstrapping conditional gans for video game level generation, 2019.

