Generation of Diverse Stages in Turn-Based Role-Playing Game using Reinforcement Learning

SangGyu Nam
School of Information Science
JAIST
Ishikawa, Japan
Email: howzen@jaist.ac.jp

Kokolo Ikeda
School of Information Science
JAIST
Ishikawa, Japan
Email: kokolo@jaist.ac.jp

***Abstract*—In this study, procedural content generation (PCG) using reinforcement learning (RL) is focused. PCG is defined as the generation of game content tailored to the defined evaluation function using RL models, which is one of the examples of PCG via machine learning. Compared to other generation content areas such as computer vision and natural language process, generative models such as variational autoencoders, PixelCNN, and generative adversarial networks exhibit some difficulties for applications to the game area because during the development of a new game, the content data used for training is typically not sufficient. Hence, RL is considered to be used as a method for PCG. In particular, the stage of turn-based RPG is selected as our research target because it comprises discrete sections, and its parameters were closely related; hence, it is a challenge to generate desirable stages, and the main goal is to generate various stages guided by the designed evaluation function. Two RL models, Deep Q-Network and Deep Deterministic Policy Gradient, respectively, are selected, and the generated stages are evaluated as 0.78 and 0.85 by our designed function, respectively. By the application of the stochastic noise policy, diverse stages are successfully obtained, and those diversities are evaluated by the parameter mse and the different number of valid strategies.**

I. Introduction

With the dramatic growth of artificial intelligence (AI), currently, AI has been used effectively in various applications. One example involves the automatic generation of content using some algorithms in games, which is referred to as procedural content generation (PCG), and it is one of the major research fields in the game area. PCG can be applied to any game. PCG was mainly investigated by the generation of stages in popular games such as scroll action games (e.g., Super Mario) [1][2], racing games [3][4], real time strategy games [5], and problems in puzzle games [6]. On the other hand, relatively few studies reported turn-based RPG (RPG), which is a well-known classic genre.

In a majority of the RPGs, players play character roles to complete the story (e.g., defeat the boss), which often requires the growth of characters. Players can make their characters stronger by receiving rewards from successive battles. If players attempt to defeat all enemies using all their resources during successive battles, then it might be difficult to defeat the boss, or if players ignore all enemies, then characters may not have sufficient strength to defeat the boss. Hence, players must decide their own strategy, such as "Win this battle safely using some resources," "Save mana or items to prepare for difficult battles," or "Retreat against difficult enemies," leading to the entertainment of the RPG.

To make available the various strategies, it is crucial to assign appropriate locations of enemies, frequency of recovery, and several parameters such as the strength of the enemies, the effectiveness of the items, and to the extent of degree to which the characters are cured, and others, which is the balance of the "Stage." In addition, in terms of fun, it is crucial to provide diverse stages, which makes players decide different strategies. With these stages, RPG can avoid the monotony, and players can feel refreshed. Hence, the research goal involves the generation of these diverse stages, leading to entertainment.

PCG has several approaches. One example is procedural content generation via machine learning (PCGML) [7]. PCGML was actively investigated in recent years. In several PCGML studies, network models use the existing game content data to learn generation. When considerable content created by human designers can be easily obtained, it might be possible to generate game content based on data distribution using generative models such as variational autoencoders (VAE) [8], PixelCNN [9], or generative adversarial network (GAN) [10]. However, with respect to PCG, it is difficult to collect sufficient content used for training data. Furthermore, it may be not guaranteed that the generated content is desired by designers, and it may not be varied as it follows the distribution of training data. Hence, generate-and-test algorithms such as search-based PCG [11] are typically used. The training data are not required in search-based PCG, but the human-defined evaluation function is needed instead, which exhibits an advantage in that each evaluation function can be tailored to the direction required by the designers or by specific players according to their skills or taste. Typically, generate-and-test algorithms generate content by the optimized evaluation or fitness value using a genetic algorithm (GA) [12]. However, GA is slightly slow in instantly providing content to various players; in addition, GA possibly generates "Similar Content Group" in principle.

Thus, reinforcement learning (RL) is considered to solve this disadvantage. Assume that the complete stage comprises n discrete sections, "State" is any stage with n or less sections, and "Action" is the generation of the next section, which is not generated thus far. When the stage is completed, then its good or bad is evaluated by the designed evaluation function, and its value is given as a "Reward." With the progress of learning, a desired stage defined by the evaluation function is generated.

978-1-7281-1884-0/19/$31.00 ©2019 IEEE

In addition, by using the stochastic noise policy, which generates noise parameters of the section according to Q values and by using a random incomplete stage as the initial state, various stages can be obtained.

II. Related Work

The structures of previous methods for PCG are approximately "generating content using some methods" and "in some cases, evaluating content and using only a good one." The generation methods are classified as follows.

- Heuristic
- Scientific models such as automaton or fractal [13]
- Reuse part of the existing data
- Machine learning generative model using training data such as GAN [2], PixelCNN, and VAE

However, unlike the generation of terrains or maps, which exert a low impact on the winning or losing of games, the generation of only RPG stages cannot guarantee that it provides satisfaction to the player as the parameters of stages affect each other in a complex manner.

A. Search-based PCG

Thus, in several cases, search-based PCG, which is one of the special cases of generate-and-test algorithms, is used. PCG involves the combination of the "selection by an evaluation function" and "generating content using any PCG methods." Among a number of the generated content, search-based PCG provides only highly evaluated content. Fig. 1 shows an example of the optimizing evaluation value and generation of a three-section stage using GA. This method exhibits three disadvantages.

- Is it possible to appropriately define a desired function that evaluates the fun of stages for humans?
- Is it possible to obtain a solution with a high evaluation value through optimization?
- Is it possible to generate solutions rapidly and diversely?

In particular, the third one is a unique concern in search-based PCG, and it makes it difficult to generate online content. That is, there is no problem in case developers generate considerable content and provide it to several unspecified players later. However, it is difficult to provide various stages, which immediately fits to specific types of players in terms of their skills or preferences. GA typically requires several evaluations to keep several solutions; moreover, such solutions tend to similarly converge depending on GA models.

B. PCGML

In several PCGML studies, network models are trained on the existing game content. Compared to search-based PCG, in PCGML, the generated content is not selected by searching the target content space by evaluation, and the content is directly generated from learned network models. PCGML also exhibits some challenges.

- Is it possible to collect sufficient data?
- Do the collected data exhibit the desired distribution?

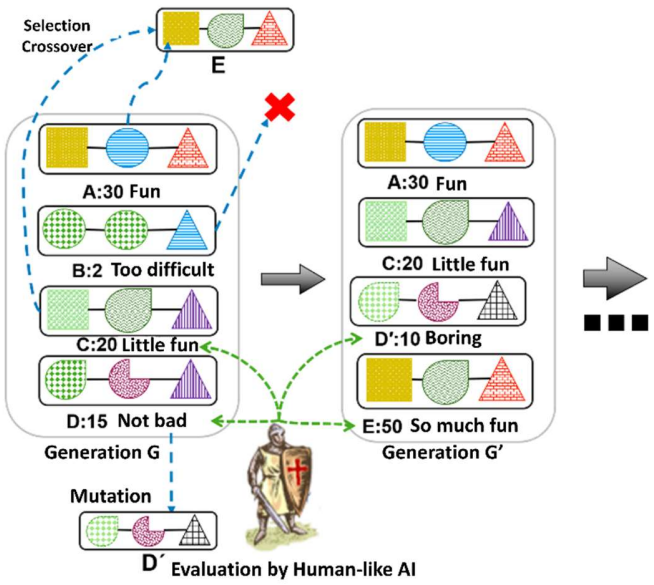

Fig. 1. Stage generation using a genetic algorithm

In case of image generation, the collected data can be simply categorized into a horse, a car, or a bird. Its goal is to make models learn the distribution of the target category. However, in the case of games, it may be crucial to sort the data by types, and it is difficult work as it involves the identification of those characteristics. Beyond this issue, the collection of sufficient data can be difficult considering that the content of different games exhibits marginal compatibility.

C. Reinforcement Learning

RL is one of the categories of machine learning that permits agents to learn the way to act in a specific situation to maximize the expected cumulative reward received from the environment. Compared to supervised learning, which is also one of the categories of machine learning, RL does not require data; instead, it does self-learning through interactions with the environment. In PCG, the generation with a co-creative agent is proposed by using RL [14][15]. In these papers, they set rewards based on the user input. The learning progress of RL is guided by the reward. Hence, it will decide the type of content that is generated. In our case, the evaluation function was directly designed, and its value as a reward was given; hence, we attempt to make our models learn as desired.

III. Turn-based RPG and Stage

Various examples of RPG include Dragon Quest, Final Fantasy, Pokémon series, or others. Even though each game comprises its own unique system, a majority of the games comprise stages consisting of battle and non-battle sections. In the battle section, player and enemy teams fight each other acting in order until one team is defeated or in case of retreat. Although each game comprises a different system, the condition of characters is inherited after the battle in a majority of the games. Thus, just winning an immediate battle is not sufficient considering future battles. This is an especially different element compared to other game genres. In non-battle sections, damaged characters can be cured or the player

can buy items and equipment to reinforce characters. For each stage, medium-and long-term goals exist, such as "take gold from weak enemies, buy equipment, save magics, and beat the boss." Hence, players need to make decisions to advance more or withdraw to town, use or save magic, and select which member can be a party member. In other words, it is desirable to generate stages that require such decisions.

A. Battle and Non-battle Sections

The battle of RPG occurs on the basis of "Turn." At every turn, each character performs a single action in order. The manner in which the order of characters is decided is different in each game; however, in a majority of games, it is basically determined by the speed parameter. All characters exhibit their own unique actions such as attack, magic, skill, and others, and these actions make each character play a unique role. As all characters exhibit a unique personality and role, these games are called role-playing games. For example, Fig. 2 shows the flow of one turn in battle.

Once all of the characters have completed their actions, then a single turn is over. The turn will be continued until one of the player teams or enemy teams is eliminated, implying that the hit point parameters of all characters are zero or retreats. Generally, player characters obtain rewards after winning a battle, e.g., strengthen characters or gold or items.

The non-battle section varies depending on the game; however, some common factors exist, such as, recovering wounds in the inn, reinforcing characters by buying weapons or armors from an equipment shop, buying recovery items from a potion shop, and acquiring items from treasure chests or others. These factors are common to a majority of RPGs.

B. Desirable Stages

Thus far, the hand-crafted stages are mainly played. However, there is an increasing number of games using automatically generated stages to provide new experiences to players. However, all parameters of sections are closely related, and the parameters affect each other; hence, the determination of parameters at random is not sufficient to make satisfying content. Thus, the following points need to be considered.

1) If the difficulty of the stage is extremely easy or difficult, e.g., a player can beat the stage with any strategy or strategies are not valid, then, the only thing remaining for a player to do is just pray, and this type of a stage is not interesting.
2) If a stage could be beaten by almost the same strategy even after the parameters of the stage are changed, then we cannot say that the game is interesting. It is better to have various strategies depending on the situations.
3) When an obvious action or a strategy is clearly effective or ineffective against the stage, we cannot say that it is fun. Stages that make a player hesitate selecting an action can entertain players.

C. Research Platform

Several commercial RPGs have rich stories, several game modes, and adventure maps. In this study, these elements are not needed as there is no interest in them; only battle-related

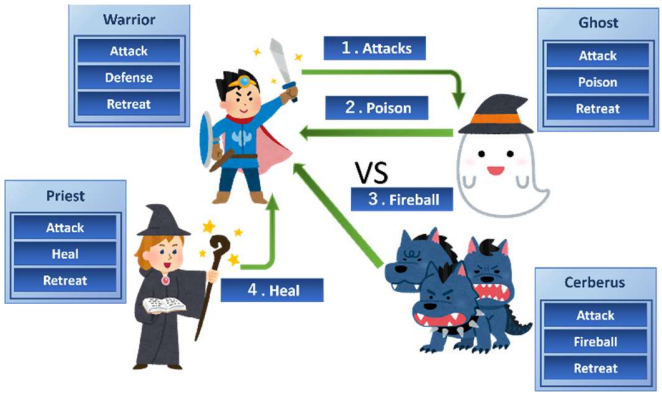

Fig. 2. Flow of one turn in the RPG battle section.

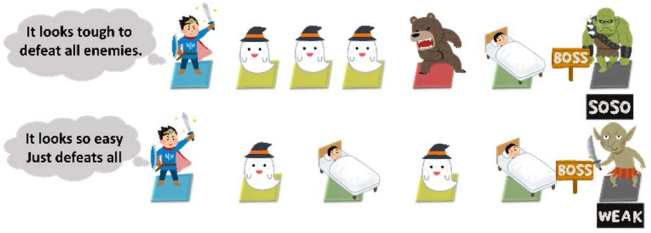

Fig. 3. Examples of various stage structures.

parts, battle section, and non-battle recovery section is focused. It is a common practice to use a restricted platform for research. However, an appropriate platform is not found; hence, the following simplified platform is used on the basis of the original platform implemented by our group.

- **Stage composition**: Stage exhibits a unidirectional single-row n-section structure (Fig. 3). Each section is one of a battle, recovery, and boss battle.
- **Character type**: Both player and enemy teams consist of a single character, and three characters of a player, an enemy, and a boss, respectively, are present. Enemy and boss exhibit different ranges of available parameters, wherein boss exhibits a slightly higher, comprising Hit Point (HP), Attack (ATK), and speed.
- **Speed**: The speed of the players is greater than that of the enemy characters.
- **Action**: Two available actions, an attack and a retreat, respectively, are available. When a character executes attack on an enemy, the HP parameter of a target is then reduced by the ATK parameter of the executer. Retreat is only available for the player, and it ends the battle.
- **Battle Result**: When a player retreats from the battle, the player character loses 15% of HP and proceeds to the next section, and if the player wins the battle, the character gains 10% of ATK.

D. Strategy

Strategies can be interpreted in various methods depending on where the phrase is used. In addition, in computer games, more than two strategies might exist, e.g., the preservation of an item in an RPG game or the order of construction buildings in StarCraft. In this study, the decision when facing a battle or a non-battle section in RPG was

focused. In these games, players can decide to challenge or avoid the battle event; in addition, they can buy a weapon or sell items in the shop. The combination of these decisions is considered as the strategy.

Basically, the stage can be evaluated by the test play of human-like AI that plays strategies acquired from a human play. However, the implementation of human-like AI itself is a quite large, important research issue. As our simplified platform allows only two actions, the strategies also can be simplified. If the stage consists of m battles, then 2^{m-1} possible strategies exist because retreating from the boss is meaningless. By defining the simple action space, the entire strategy space can be searched. Hence, there is no need of human-like AI in this study.

IV. Approach

First, the elements of Markov decision process (MDP) was defined at the stage generation. We define all stages including the incomplete stage as the "State," manipulating parameters of an incomplete stage to the complete stage as the "Action," the completed stage as the "Goal State," and the evaluation function as the "Reward." Against the complete n-section stage, the online generation of various and desirable complete stages is expected by using an incomplete random initial stage, the section size of which is under n and the stochastic noise policy.

As RL methods for PCG, a Deep Q-Network (DQN) was first selected [16], and some limitations in PCG were observed; hence, the deep deterministic policy gradient (DDPG) is selected next [17]. DQN is a well-known method that obtains good results on several games, which comprise a discrete action space and large state space. However, as the DQN action space is suitable for a discrete and lower dimensional action space, it is difficult to generate several parameters at once. In addition, if sections of the stage have a wide range of parameters, it is difficult for the discrete action to cover all parameter ranges. Hence, DDPG is attempted, which can deal with a high-dimensional continuous action space.

Several issues need to be considered, e.g., formalization of the MDP of RL, representation of stages, and the generalization of state space and action.

A. MDP Formulation

In this section, the methods utilized from the MDP formulation are explained. First, an evaluation value $f(s^*)$ against the complete stage $s^* \in S^*$ is present, which is acquired from the designed evaluation function. Designing how to evaluate the stages has difficult issues such as "What is the difficulty?," "What is fun?," "Does a computer know the weakness of human?," or others.

Next, let the whole stage set S, including incomplete stages be a "state space" and generating some parameters (make the incomplete stage closer to the complete stage) to the incomplete stage be an "Action" $a \in A$. The episode of MDP terminates when an incomplete stage reaches the complete stage $s^* \in S^*$, and the evaluation function value $f(s^*) \in R$ is given as a reward (Fig. 4). With such an MDP formulation, the stage generation problem can be changed to the RL of the reward maximization problem. If the Q value against arbitrary

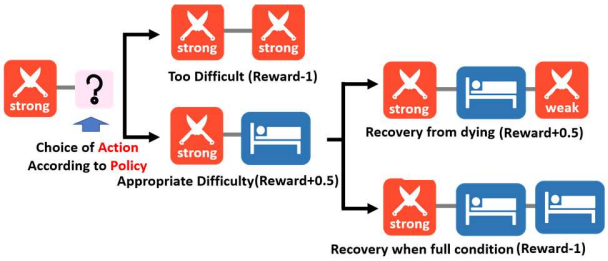

Fig. 4. MDP process of generating stages

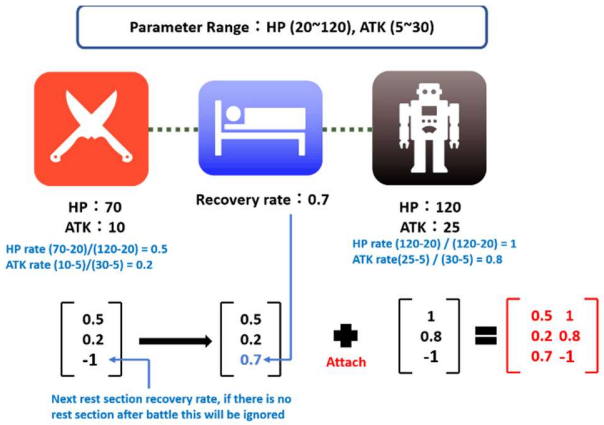

Fig. 5. Example of converting the battle-recovery-boss stage to the stage matrix

s and a are appropriately learnt, then it is possible to make various stages, as well as a sufficient stage, by selecting the action with the best Q or second-best Q.

B. Expression of a Stage Matrix

The composition of the section type in the stage can be dynamic; however, in this study, the series of the section types was fixed. Let the n size stage consists of m battle sections, including one boss section and n-m recovery sections. Let the battle sections have P_b parameters, and the recovery section have P_r parameters. Then, the row size of the stage matrix size is $P_b + P_b$, and the column is m. The whole size of the stage matrix is $(P_b + P_b) \times m$, and one column includes the information of one battle section and one next recovery section. The details are as follows.

- Battle (Boss) section

Enemy characters have HP and ATK parameters. Typically, zthe ranges of the parameter values are already set at the minimum and maximum in the developer's mind like HP is (Hpmin – Hpmax), and ATK is (ATKmin – ATKmax). Hence, it is possible to calculate the rate for each parameter.
HP rate = (HP value – Hpmin)/(Hpmax – Hpmin)
ATK rate = (ATK value – ATKmin)/(ATKmax – ATKmin), and let these values be the first and second values of the section vector, respectively.

- Recovery section

There are occasional recovery sections that make the characters recover some portion of the HP. The recovery rate is the third element of the section vector. If the recovery

section does not exist after a battle section, then this value is ignored.

Fig. 5 shows an example of the three-section stage matrix, comprising one battle, one recovery, and one boss when the HP ranges from 20 to 120, and the attack ranges from 5 to 30.

C. Selection of an Action

Next, the implementation of the action, which is the output of DQN or DDPG, is explained. As the action spaces of DQN and DDPG are different, they are explained separately.

First, the DQN of stage generation was implemented. The DQN action is one of the discrete values ranging from 0 to 100, implying a parameter range of 0% to 100% in 1% increments. As DQN handles a low-dimensional action space, it cannot help, but it only generates a single parameter per single action; otherwise, the discrete size of the action space will bigger. For example, if a state is generated up to three battles and the recovery section, then the next action is the hp parameter rate of the 4th enemy character, followed by the attack parameter and next recovery rate. This method exhibits the following limitations.

- The action occurs on different parameters. It is slightly unnatural that different types of action occur in order.
- It is difficult to generate section parameters at once. For example, if we want to generate the whole stage vector at once, an action size of 100^3 is required.
- Some parameter values are possibly not covered, such as 0.5% of HP, 55.24% of ATK, and 42.103% of ATK values in the parameter ranges.
- Almost the same values of parameters such as 54%, 55% are treated as completely different. Hence, learning might be ineffective.

In contrast to DQN, DDPG is expected to have the following advantages.

- The output of DDPG could be the real number of p dimensions. Hence, all p number of parameters (one column) can be generated at once. In addition, all values of the parameters can be generated, and the close value of two actions is treated to be similar.

D. Evaluation Function

As mentioned in Section III-C, it is a difficult job to evaluate the stage as various factors need to be considered. In this study, as the first step, a difficulty-based evaluation function was defined. As only 2^{m-1} strategies existed, the difficulty of the stage was confirmed by searching all strategies. Concretely, the rate of the number of several strategies that can defeat the boss among all strategies can be found, referred to as the "winning rate." If the winning rate is 90%, the stage can be beaten with almost any strategy. If the winning rate is 5%, the player needs to carefully select an action or it cannot be beaten. In this study, the target winning rate was assigned to be 30%: The closer to the target rate, the higher the evaluation. In addition to the winning rate, some evaluation criteria that can affect the entertainment of RPG, e.g., a bonus when a dying character is healed from the recovery section, a penalty when enemies in the early stage are extremely strong, when later enemies are extremely weak, or a penalty when the stage can be beaten by a monotonous strategy like only attacks, were also added. Designing an evaluation function that shows how fun the stage is in numerical value itself is an important research subject. However, in this study, the generation method was mainly focused; hence, a relatively simple designed evaluation function is used.

E. Random Initial Stage and Stochastic Noise Policy

In RL, policy basically decides an agent's action at a given state according to the Q value, indicating that when the same state is given and there is no randomness, the result is always the same. Hence, if the same incomplete stage is given, then the same stage is generated; hence, diverse stages cannot be obtained. Hence, in this study, the first two column parameters are selected to be completely random, and let the remaining parameters be learnt and generated by RL and the stochastic noise policy is used, which does not select the action by the best Q action (DQN) or by the actor (DDPG). The second-best action is selected on the basis of the Q distribution. In DQN, the second-best action is easily obtained by selecting the second- or third-highest action. However, in the case of DDPG, it is impossible to check all actions and their Q values. Hence, all parameters are roughly checked, and the distribution of Q values is investigated to understand the relationship between Q values and the actual reward distribution. From this relationship, the stochastic noise policy is designed (Section VII. A)

V. Preliminary Experiment: Supervised Learning of Evaluation Function

It is a precondition that our proposed stage generation model can estimate the Q value against any S, including the complete stage. It is possible to evaluate the complete stage s* by reward, but in the case of the incomplete stage, there is no method to evaluate its goodness; hence, it can only be estimated using Q value from some approximate model. As each section is closely related to each other, whether it can be appropriately estimated by the model is doubtful. Therefore, as a preliminary experiment, supervised learning is performed, which takes the complete stage and evaluation value as the input and output, respectively. If good estimation accuracy cannot be obtained even for the complete stage, then it is impossible to accurately learn the Q value in RL models.

A. Experimental Setup

The stage comprises 9 sections: 3battles-recovery-3battles-recovery-boss. As there are seven battle sections, 64 strategies exist, and the input size is $3 \times 7 = 21$. The network has 32 nodes of two convolution layers, with one average pooling, and 256 nodes of seven completely connected layers. The activation function is ReLu, and 12000 train stages and 4000 validation stages are generated to obtain a uniform distribution of the evaluation value. In this preliminary experiment, the evaluation function only uses the winning rate.

B. Result

Fig. 6 shows the result. The validation loss is almost 0.0055, indicative of an estimation error of 7–8%; in terms of strategy, there are five error strategies. It cannot be deemed as perfect precision, but the evaluation values of the stage can be thought to be estimated to some degree.

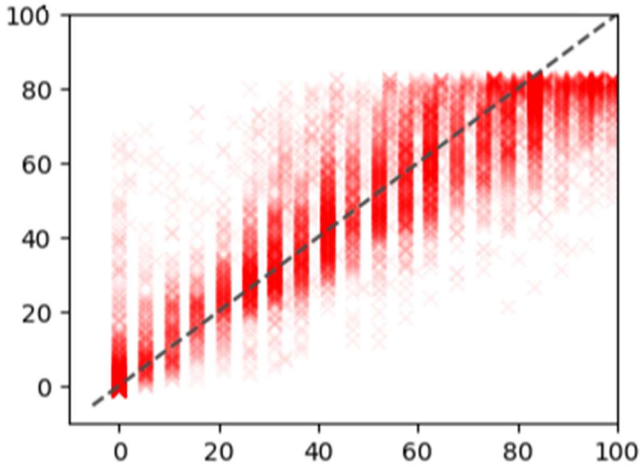

Fig. 6. Relationship between the expected evaluation value (vertical axis) and actual evaluation value (horizontal axis)

VI. Experiment: Stage Generation

At least in our research platform, the possibility of estimating the Q value from the result of V was judged; hence, experiments are performed to generate stages using DQN and DDPG.

The most contrasting difference between DDPG and DQN is the action space. The output of DDPG is a three-dimensional real value between 0 and 1. Hence, against the 3×7 stage matrix which first two-columns are already generated, DDPG only needs five actions to complete the stage, while DQN needs 14 actions.

A. Experimental Setup

The stage configuration is the same as the experiment in V. The DQN network has 36–64 nodes of two convolution layers and one average polling and 128–256–256–256 nodes of four completely connected layers. The activation function is ReLu. The whole episode number is 20000, the experience memory size is 100000, the learning rate is 0.00025, the batch size 128, gamma is 0.9, and the frequency of the target network is 2000.

There are two networks in DDPG: actor and critic. The actor network has 128–256–512 nodes of convolution layers and 128–256–256–512 nodes of completely connected layers. The critic network has 128–128–256 nodes of convolution layers and 128–256–256–512 nodes of completely connected layers. The learning rate is 5×10^{-5} for actor and 5×10^{-4} for critic, the batch size is 64, the TAU for the soft update is 0.001, the episode number is 100000, and the experience memory size is 300000.

B. Result

Stages could be evaluated at an average of 50 episodes at ~0.78 (Fig. 7) and 0.85 (Fig. 8) for each DQN and DDPG. The total episode is apparently different, as well as the setup; however, several settings were attempted, and the limitation of DQN was ~0.75. It is possible to generate good stages evaluated by our designed function. Of course, we cannot say that the generated stages are fun to play. Fig. 9 shows an example of the two stages generated at random (upper) and

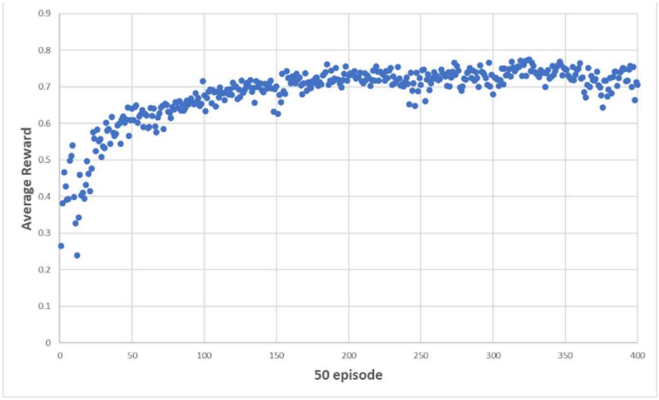

Fig. 7. Average reward for 50 episodes (total 20000 episodes) of stage generation using DQN

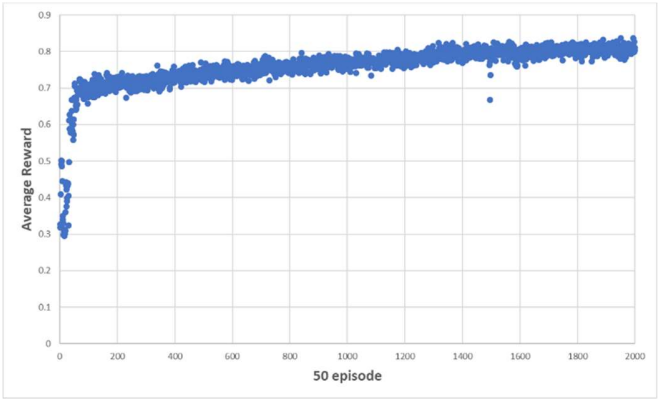

Fig. 8. Average reward for 50 episodes (total 100000 episodes) of stage generation using DDPG

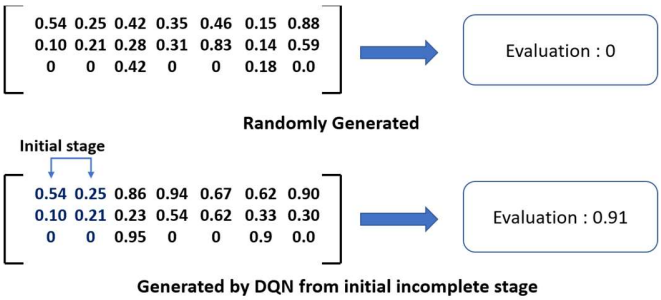

Fig. 9. An example of two stages, where each column represents section parameters, and hit point, attack, and recovery rate from the top represent the parameters (upper: randomly generated, bottom: generated by DQN)

DQN (bottom). The upper one is so easy that any strategy is valid; hence, its evaluation value is 0. The bottom one is generated from the first two sections (two battles), which is a part of the upper stage, indicating that the parameters of the enemies are raised; hence, the stage becomes more difficult than random one.

VII. Experiment: Diverse Stage Generation

From Section VI, it is possible to generate the stage with a high evaluation value. However, our main goal is to make diverse and desired stages. As DDPG gave a better result, an

experiment VII was performed only with DDPG. To make diverse stages in DDPG, a relationship between the Q value and evaluation value against all approximate parameters is first investigated. A specific one action column (noise column) of the stage matrix is generated by the increase of all of the approximate parameters (noise parameter) from 0% to 100% by 1% for each HP and ATK. Let the recovery rate be the same as that of the actor. For example, if the action of actor policy is [0.3, 0.04, 0.5], which means HP 30%, ATK 4%, 50%, then, we investigate from [0, 0, 0.5], [0, 0.01, 0.5], [0, 0.02, 0.5] … to [1, 1, 0.5]). After generating one specific stage column by this noise parameter, the other incomplete part is generated by the actor policy. For example, if the second-stage column is generated by the noise parameter, then the remaining parts are generated by the actor policy (Fig. 10, upper left). Fig. 10 shows the result obtained from when noise column is the second column to the last column. We would like to investigate the fact that if we select a parameter according to the high Q value and other parts are generated by the actor policy, would it be possible to obtain a sufficiently good stage? If so, then we can obtain the justification for selecting the action according to the Q value. The reward and Q value are somewhat directly proportional (Fig. 10). Hence, if a high Q value action is selected, then the generated stage is likely to be a highly evaluated one.

With the progress of learning, less exploration will occur, and then the reward of the RL model will be increased; instead, it will become vulnerable to an unexpected action. In a majority of the RL cases, it is not an issue as many of RL's goal is to obtain the highest cumulative rewards. However, in the case of this study, the goal is to generate various stages, which are the second-best stage by using an unexpected noise action. Vulnerable to the unexpected action occurs because of the recovery sections that can make a player full condition. Even though parameters of incomplete stages are quite different they are considered as the same so, actions after the recovery section will be almost same (Fig. 11). To solve this problem, our model needs to be made flexible against any unexpected action. Hence, the incomplete random initial stage of a fixed two-section size is changed to a random dynamic (1 to 6) size stage so that the model can be trained in various unexpected situations. By doing so, the network can learn against unexpected states as it can encounter dynamic states.

A. Detail of the Stochastic Noise Policy

By comparing the Q value and reward distribution, we know that if we select a noise parameter with a high Q value, then there is a high possibility that the completed stage exhibits a high evaluation value. However, there is also a small, but minor, possibility that the selection of a high Q value parameter possibly makes a low evaluated one. Making a low Q value with high evaluated stages is not an issue; however, in the opposite case, it can be a serious issue. To avoid this issue, a stochastic noise policy was designed, generating stage candidates based on the Q value and then providing a good stage. Details are as follows.

- Decide the target columns of the stochastic noise policy. The number of targets can be greater than 1.
- Check whether the currently generating column is a target of the stochastic noise policy.

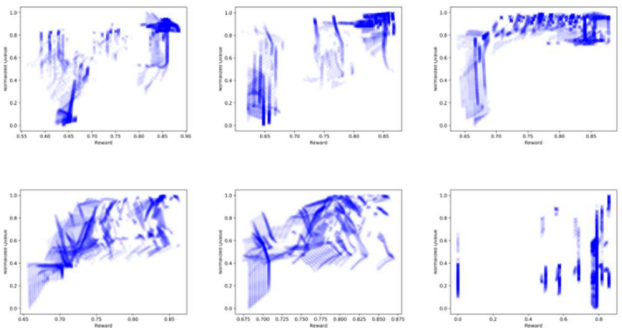

Fig. 10. Relationship between Q (vertical axis) and reward (Horizontal axis)

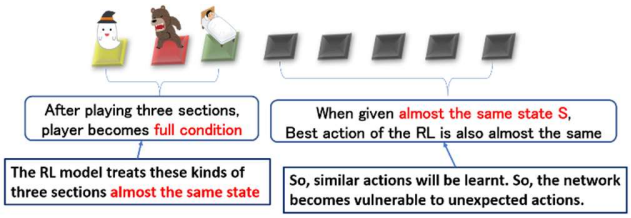

Fig. 11. Problems occurring at the stage generation using RL.

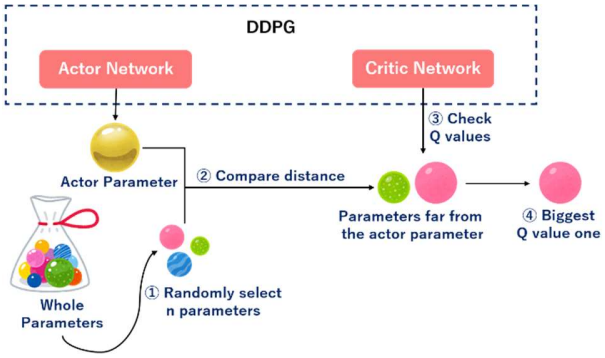

Fig. 12. Flow of selecting the noised action in the stochastic noise policy

- If not, the column is generated by the actor policy.
- If so, make n random parameters. Among them, parameters are discarded if the distance between a parameter and an actor parameter is less than the length d as it indicates that they are close to each other.
- Generate a column with the highest Q value action that satisfies the above conditions (Fig. 12).
- Repeat the entire process m times.
- Among the m stages, provide a good one.

B. Result

Diversity was evaluated by the extent of difference among the m stages and the stage generated by the actor. The degree of difference was evaluated by the parameter mse, which involves the calculation of the mean square error of the two stage matrixes and by the number of different valid strategies. Fig. 13 shows the result obtained for the average mse and average of the different numbers of valid strategies and average reward. The higher the n, the higher the evaluation

Fig. 13. Average reward and the average number of different valid strategies as well as the average parameter mse of stages generated by the stochastic noise policy when n = 5, 10, 20, and 50; m is 50; and d is 0.2. The target noise column are 1st and 6th.

Fig. 14. Stages generated by the actor policy and stochastic noise policy (sections 3, 4, 7, and 8 are noised sections). Actor one is evaluated as 0.928661, the other is evaluated as 0.894968. The stage parameter mse between the two stages is 0.3525, and the different numbers of valid strategies is 5.

value and the lower the parameter mse and different strategy counts. A high n indicated that there are high chances to get a parameter with high Q value; this result revealed that it is likely to have a high evaluation value. In other words, diversity and good quality are inversely proportional to each other in our designed game and evaluation function. Fig. 14 shows the two stages: above stage is generated by an actor policy and the other is generated by the stochastic noise policy.

VIII. CONCLUSION AND FUTURE WORK

In this study, to generate various and desired RPG stages, procedural content generation using reinforcement learning and stochastic noise parameters is introduced. Two RL algorithms, DQN and DDPG, respectively, are implemented. The results revealed that a discrete action space of DQN exhibits some limitations for PCG. However, in terms of the target, parameters have small ranges and a low dimension; DQN exhibits advantages in that it can easily find the second-best action. Compared to DQN, DDPG can possibly cope with any range of parameters. In addition, results revealed that the stages generated by DDPG are slightly better than those generated by DQN.

In addition, various stages are generated by using an incomplete random initial stage. As the result is changed by the initial stage, the stochastic noise policy is also employed. Through experiments, the quality and diversity are inversely proportional in our target environment.

However, whether users actually feel various to those generated stages are not evaluated, as well as whether the evaluation function actually reflects the entertainment of users. These factors will be confirmed by performing subject experiments. Moreover, our stochastic noise policy is quite a simple policy, and we cannot say that it is the best method to make the second-best stage. Hence, we intend to focus on improving the efficiency of the stochastic noise policy. Moreover, there are more challenges when this method is applied in actual commercial RPG; hence, this method will be applied in a more complicated environment than simplified one in this paper, and problems will be attempted to be solved.

ACKNOWLEDGEMENT

This work was supported by JSPS KAKENHI Grant Number 18H03347, 17K00506.

REFERENCES

[1] A. Summerville, M. Mateas, "Super Mario as a string: Platformer level generation via LSTMs," Proc. 1st Int. Joint Conf. DiGRA/FDG, 2016.

[2] V. Volz, *et al.* "Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network," in GECCO, 2018.

[3] D. Loiacono, L. Cardamone, P.-L. Lanzi, "Automatic track generation for high-end racing games using evolutionary computation," IEEE Trans. Comput. Intell. AI Games, 3(3):245–259, 2011.

[4] J. Togelius, R. De Nardi, and S. M. Lucas, "Towards automatic personalised content creation in racing games," in Proc. IEEE Symp. Comput. Intell. Games, 2007, pp. 252–259.

[5] T. Mahlmann, J. Togelius, and G. N. Yannakakis, "Spicing upmap generation," in EvoApplications, vol. 7248. Springer, 224-233, 2012.

[6] A. Liapis, C. Holmgard, G. N. Yannakakis, J. Togelius, "Procedural personas as critics for dungeon generation", European Conference on the Applications of Evolutionary Computation, pp. 331-343, 2015.

[7] A. Summerville *et al*. "Procedural Content Generation via Machine Learning (PCGML)," IEEE Transactions on Games, 10:257–270, 2018.

[8] Diederik P Kingma, Welling Max, "Auto-encoding variational bayes," arXiv preprint arXiv:1312.6114, 2013.

[9] A. van den Oord, *et al.* "Conditional image generation with pixelcnn decoders," In Advances in Neural Information Processing Systems, pp. 4790–4798, 2016.

[10] I. Goodfellow, *et al.* "Generative adversarial nets," In Advances in Neural Information Processing Systems, pp. 2672–2680, 2014.

[11] J. Togelius, G. N. Yannakakis, K. Stanley, C. Browne, "Search-based Procedural Content Generation: A Taxonomy and Survey," IEEE Trans. Comput. Intell. AI Games, (99):1–1, 2011.

[12] M. Stephenson, J. Renz, "Procedural generation of complex stable structures for angry birds levels," in 2016 IEEE Conference on Computational Intelligence and Games, pp. 1–8, 2016.

[13] N. Shaker, J. Togelius, MJ. Nelson, "Fractals, noise and agents with applications to landscapes," In Procedural Content Generation in Games. 57–72, 2016

[14] M. Guzdial, N. Liao, M. Riedl, "Co-creative level design via machine learning," arXiv preprint arXiv:1809.09420, 2018.

[15] M. Guzdial, *et al.* "Friend, Collaborator, Student, Manager: How Design of an AI-Driven Game Level Editor Affects Creators," arXiv preprint arXiv:1901.06417, 2019.

[16] Mnih, V. *et al.* "Human-level control through deep reinforcement learning," Nature 518, 529–533, 2015.

[17] T. Lillicrap, *et al.* "Continuous control with deep reinforcement learning," arXiv preprint arXiv:1509.02971, 2015