JAIST Repository
https://dspace.jaist.ac.jp/

Title	Generation of Game Stages with Quality and Diversity by Reinforcement Learning in Turn-based RPG
Author(s)	Nam, SangGyu; Hsueh, Chu-Hsuan; Ikeda, Kokolo
Citation	IEEE Transactions on Games (ToG), 14(3): 488-501
Issue Date	2022-09
Type	Journal Article
Text version	author
URL	http://hdl.handle.net/10119/18245
Rights	This is the author's version of the work. Copyright (C) 2022 IEEE. IEEE Transactions on Games, 14 (3), 2022, 488-501. DOI: 10.1109/TG.2021.3113313. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Description	

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

Generation of Game Stages with Quality and Diversity by Reinforcement Learning in Turn-based RPG

Sang-Gyu Nam, Chu-Hsuan Hsueh, Kokolo Ikeda

***Abstract*—Many recent studies in procedural content generation (PCG) are based on machine learning. One of the promising approaches is generative models, which have shown impressive results in generating new pictures and videos from existing ones. However, it is usually costly to collect sufficient content for training on PCG. To address this issue, we consider reinforcement learning (RL), which does not need to collect training data in advance but learns from its interaction with an environment. In this work, RL agents are trained to generate stages, which we define as series of events in turn-based role playing games (RPG). It is a challenging task since several events in a stage are usually highly correlated to each other. We first formulate the stage generation problem into a Markov decision process. A hand-crafted evaluation function, which simulates players' enjoyment, is defined to evaluate generated stages. Two RL algorithms are selected in the experiments, which are deep Q-network (DQN) for discrete action space and deep deterministic policy gradient (DDPG) for continuous action space. The generated stages from both models receive evaluation values indicating good quality. To solve the delayed reward problem and further improve the quality of the stages, we employ virtual simulations to give rewards to intermediate actions and get stages with higher average scores. In addition, we introduce noise to avoid generating similar stages while trying to keep the quality as high as possible. The proposed methods succeed in generating good and diverse stages.**

***Index Terms*—Reinforcement learning, procedural content generation, turn-based rpg, machine learning, quality, diversity.**

I. Introduction

PROCEDURAL content generation (PCG) has arisen as one major research field in games. PCG in games implies the generation of game content with algorithms. Many elements in games can be the target of content such as levels [1], textures [2], puzzles [3], items [4], and NPCs [5]. Although generating content itself is the primary goal of PCG, there are other usages like training AI players in various environments to increase their generality [6].

Research in PCG has been conducted on many different kinds of games, which all have their own unique challenges. The majority of studies about PCG are mainly focused on a subset of game genres such as platformer games (e.g., Super Mario [7]) [1][8], racing games [9][10], and problems in puzzle games [3]. However, a well-known classic genre, turn-based RPG[1], is relatively less studied.

In most turn-based RPGs, players control characters aiming to reach the final goal (e.g., defeating the final boss), though different games may have different story settings. For example, players can make their characters stronger by obtaining rewards from battles with monsters, and those procedures occur successively. Such series of events players may encounter during the game are seen as a stage of RPGs in this paper. In well-designed and challenging RPGs like Darkest Dungeon [11] or the Mysterious Dungeon series [12], if players attempt to defeat all consecutive monsters using all of their resources, then they may fail to defeat the boss. Alternatively, if players ignore all enemies, then the characters may not have adequate strength to defeat the boss. Because of these unique features, under many circumstances, players need to devise strategies, such as winning a tough battle using all resources or saving items for more important battles later. These lead to the entertainment of turn-based RPGs.

Stages of turn-based RPGs should be well designed so that players need to consider their strategies carefully, which makes them feel challenging and entertaining. It is crucial to make a balance between events in the stages, such as the locations, frequencies, and statuses of enemies, the locations and effectiveness of recovery points, and the effects of items. Game designers have employed constructive PCG methods (usually hand-designed rules) to generate stages in many commercial turn-based RPGs, which gives designers a high level of control on the generated content. However, it requires experts' dedication to thoroughly create rules or decide parameters, which still cannot guarantee adequate game balance. For example, Disgaea [13] is one of the famous turn-based RPGs that have parameters in a wide range. Despite that, many enemies in later phases of the game can be defeated by only one attack as player characters get stronger rapidly compared to enemies. Providing diverse stages to players is another aspect to consider for entertainment. Players need to develop different strategies under different circumstances, which usually makes play enjoyable, especially in turn-based RPGs. Constructive PCG methods also have potential issues about lacking diversity in that players may somehow find patterns in the stages. In this research, we aim at generating not

The authors are with the School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa, Japan, e-mail: (howzen@jaist.ac.jp; hsuehch@jaist.ac.jp; kokolo@jaist.ac.jp)

[1]The definition of turn-based PRG may vary from wide to narrow. In this work, we consider games like Wizardry and Darkest Dungeon and will present a simplified implementation in Section III.

only high-quality stages of turn-based RPGs but also diverse ones based on PCG.

Researchers have tried several approaches for PCG. One example is procedural content generation via machine learning (PCGML) [14]. In most PCGML studies, models learn how to generate content using existing content. When it is easy to obtain substantial content created by human designers, it might be possible to generate game content based on the distribution of prepared data using generative models such as variational autoencoders (VAE) [15], PixelCNN [16], or generative adversarial networks (GAN) [17]. However, concerning actual game development, it is not easy to collect a sufficient amount of content for training. Thus, it may result in generating similar content to existing one and lacking diversity.

There is another approach called search-based PCG [18]. The optimization systems in search-based PCG mainly consist of two parts, the generator and the evaluator. Usually, game content is represented by parameters and generated by repeating the following processes, 1) the evaluator grades content by one or a vector of real numbers instead of a simple acceptance or rejection, and 2) the generator aims to find better parameters. Search-based PCG does not require training data; instead, it requires evaluation functions, usually designed by humans. Designers can tailor the evaluation functions to their preference in games, or to some specific players' skills or taste. Typically, search-based PCG generates content by optimizing the evaluation values, and in many cases, the optimization is based on genetic algorithms (GA). Some efforts may be required to enable GA to generate diverse content [19].

Another very recent approach is reinforcement learning (RL) [20][21][22]. The work by Khalifa et al. [20] and by us [21][22] was undertaken independently, and to our best knowledge, ours is the first to train RL agents to generate game levels/stages from scratch. The main reason why we introduce RL is that there is no need to collect data and it can do online generation with the potential of getting diverse content. The general idea on transforming levels/stages generation into an RL problem in both works are the same. In more detail, both formulate the problem into Markov decision processes, where states are generated levels/stages, actions are to modify the levels/stages, and rewards are the degree of how good the generated levels/stages are. Khalifa et al. [20] put more emphases on the generality of the method and on generating playable games, while we focus more on balance between events (i.e., quality) in stages of turn-based RPGs and the diversity of generated stages. In our design, each stage comprises n events, and the generation of a stage means deciding the n events in sequence. An evaluation function is designed to evaluate the completed stages and give rewards. In addition to generating good stages, we also introduce a noise selection that selects good but different actions from the learned policy on purpose in order to generate diverse stages.

This paper is extended from our previous work [22], where we further introduce virtual simulations for improving the quality of generated stages (Sections IV-C and V-A) and conduct more thorough studies on the relation between noise and the diversity of stages (Section V-C). The structure of the rest of this paper is as follows. Section II describes the background of our work. Section III introduces our target turn-based RPG. Section IV presents our approach to generate stages by RL. Section V then shows the experiment results. Finally, Section IV includes the conclusions and discussions on future works.

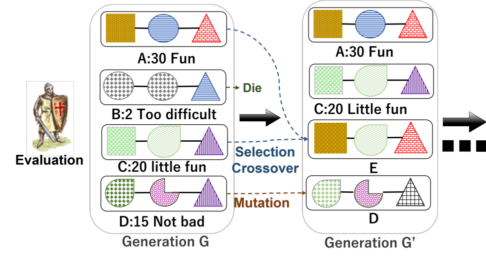

Fig. 1. Generation of stages using GA. Each stage is represented as an individual containing three discrete events. An evaluator assesses individuals (A, B, C, D) in generation G, and good individuals (A, C) are selected. Through crossover and mutation operations, better genes are passed to the next generation G'.

II. Background

In this section, we first give an overview on PCG. Next, Sections II-B and II-C introduce research on search-based PCG and PCGML, respectively, which are representative PCG approaches. As a new approach, we adopt reinforcement learning, where the general concepts of reinforcement learning are reviewed in Section II-D.

A. PCG Overview

PCG [19] creates game content algorithmically. The generation process can happen upon players' demands (online) or during game development (offline), where the time constraint usually makes the former more challenging. For both online and offline generation, the content should have high quality. In addition, diversity [23][24] is an important factor to keep players' long-term enjoyment in terms of freshness. Related to players' enjoyment, some studies considered perspectives such as difficulty [25] [26] [27] [28] and entertainment [29].

In addition to the above, PCG can be categorized based on various perspectives [30]. When considering methodologies based on artificial intelligence, two major groups are search-based PCG and PCG via machine learning (PCGML), which will be introduced more in the following sections.

B. Search-Based PCG

Search-based PCG [18][9][10][30] is one special case of generate-and-test algorithms that aims to generate good content efficiently, even for complex games. Usually, generate-and-test algorithms involve the repeated processes of generating content through the stochastic procedure and then filtering by evaluation functions until good content is obtained. However, when the target game is complex and some degree of quality is required, generating good content is quite tough as the probability of finding good content with random generation is relatively low whereas search-based PCG can generate good content in complex games [31][9][10]. Search-based PCG

mainly applies genetic algorithms (GAs), which keep evolving existing content to obtain content with better evaluation values. Fig. 1 shows an example of the generation of stages in a turn-based RPG using GA. By repeating the operations of selection, crossover, and mutation, the GA aims to generate better stages from previously generated ones. Search-based PCG can also achieve adaptive generation according to the design of evaluation functions. For example, Togelius et al. [9] used entertainment features from player logs and optimize fitness to specific players.

While search-based PCG is efficient compared to naive generate-and-test algorithms, still, it has two potential issues, the diversity of content and online generation. Since GAs tend to generate similar individuals, search-based PCG may have issues about generating diverse content. Loiacono et al. [10] tried to generate diverse content (tracks in racing games) based on Togelius et al.'s work [9]. Gravina et al. [23] proposed a new concept, PCG through quality-diversity, as a subset of search-based PCG that aims to maintain both the quality and diversity of generated content. For example, Alvarez et al. [32] used the MAP-Elites algorithm to generate designer-interactive dungeons. No matter whether diversity is required, search-based PCG needs a large number of evaluations, in other words, trials and errors. Thus, online generation may be difficult, especially when the evaluations are slow.

C. PCGML

PCGML [14] is a framework for generating content using machine learning. The generation models learn how to generate content using existing content as training data. Different from search-based PCG, PCGML generates content directly from the trained models.

PCGML has been realized by many different approaches and applied to generate various kinds of game content, mainly game levels. Summerville and Mateas [1] generated Mario levels by a kind of neural network called long short-term memory (LSTM), known to be good at predicting the next item in a sequence. They represented the 2-dimension levels as strings, where each character stood for a tile on the map. Given a seed sequence (i.e., several pre-assigned tiles), the LSTM generated a level tile-by-tile until reaching an end-of-level terminal symbol. Lee et al. [33] trained convolutional neural networks to predict resource locations in StarCraft II maps, which was considered a potentially useful tool for map designers. Summerville et al. [34] learned the room-to-room structures of Zelda dungeons by Bayesian networks and generated new levels that have similar statistical properties to given levels. Generative adversarial networks (GANs) and variational autoencoder (VAE) have become popular approaches in recent years, which successfully generated similar but new images from input datasets. The former has been adopted to PCG for generating Mario levels [8] [35], Zelda levels [36], Doom levels [37], and an educational game for middle school students [38]. The latter has been adopted to PCG for generating Mario levels [39] [40], lode runner levels [41] and levels of six platformer games [42].

PCGML may have issues such as not having enough data for learning, determining what features should the generated content use and what kind of data should be collected. For example, if the turn-based RPG stages are the target, it is difficult to say whether the collected stage data are associated with appropriate difficulty, or whether the stages require specific strategy patterns. It means that gathering data involves grasping the characteristics of the content, so it is difficult and expensive, and noise is likely to be included. In addition to this problem, it is hard for the content of other games to be reused for the target games because the content of different games has a different structure. These are some of the issues that should be considered when applying PCGML.

D. Reinforcement Learning

We adopt reinforcement learning (RL) as a new approach aiming to solve the above difficulties. For RL problems, the environments are usually modeled by Markov decision processes (MDP), a mathematical formulation widely employed when studying optimization problems. MDP is usually represented by (S, A, P, R, γ) where S is state space, A is action space, $P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the probability of transition from s to s' by performing action a at time t, $R_a(s, s')$ is the immediate reward received from the transition, and $\gamma \in [0, 1]$ is the discount factor which shows how much the future cumulative rewards is considered compared to the immediate rewards. Every next state s' depends only on the current state s and the action a. The state transition is independent of past states, which satisfies the Markov property.

Different from supervised learning and unsupervised learning, RL does not require training data but does self-learning by interacting with the environment. When all state-action pairs can be listed, whose Q-values can be stored in a table, it is guaranteed that the optimal policy can be learned given sufficient time. However, this condition is difficult to satisfy in most of the real-world problems. Thus, function approximators such as convolutional neural networks (CNNs) are used to approximate the Q-values. Mnih et al. [43] succeeded in creating AI that is stronger than humans in about half of the 49 tested Atari2600 games by the proposed deep Q-network.

Some important features of RL for solving problems are summarized as follows.

- The problems should be able to be modeled by MDP.
- Reward functions should be carefully designed so that the desired policies can be obtained when the rewards are maximized.
- States can be represented in many ways, and the representation is important since it influences the computational complexity of the problem.
- Depending on the RL models, outputs can be stochastic or deterministic.
- Learning usually requires time. However, once the learning is done, selecting actions is often fast.

Guzdial et al. [44] and Sheffield and Shah [45] formulized level generation as MDP, though their methods were based on supervised learning. As an independent work of our paper, Khalifa et al. [20] used RL for generating game levels. They designed three kinds of representations for two-dimensional game levels and applied them to three games. Some more comparisons with our design will be made in Section IV.

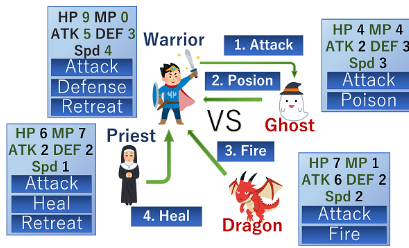

Fig. 2. An example turn-based RPG battle event and the flow of one turn. The warrior with the highest speed first attacks the ghost. The next fastest ghost performs a poison action on the warrior. Next, the dragon uses a fire skill on the warrior. Finally, the priest heals the damaged warrior.

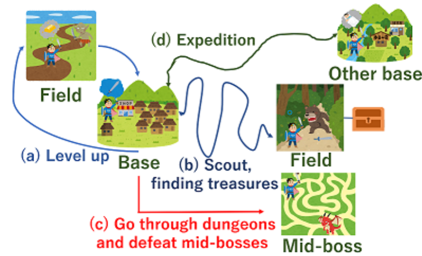

Fig. 3. Four courses in an example turn-based RPG. (a) Base → Level up, get items and gold in the field → Return to the base, (b) Base → Explore the field or dungeon → Scout or pick up treasure, (c) Base → Explore the dungeon preserving items → Defeat the mid-boss, (d) Base → Move to field → Move to a new base town.

III. Turn-based RPG and Our Target Problem

In this research, we propose a research platform for turn-based RPG and aim to generate stages in the platform. Most turn-based RPGs have unique systems but have several common elements. For example, players need to grow their characters and finally defeat the boss. A stage usually consists of several events, which can be roughly divided into battle events and non-battle events. The arrangement of events greatly influences the enjoyment of playing. Section III-A first introduces battle and non-battle events, and then Section III-B discusses elements of desirable stages. A summary of our turn-based RPG platform is presented in Section III-C.

A. Battle and Non-battle Events

In a battle event, a player's team and an enemy's team fight with each other until one team defeats the other team or flees from the battle. In most turn-based RPGs, the status of the characters is preserved after battles. For example, damage received during a battle is not recovered after the end of the battle. Thus, selecting a short-term strategy like just winning the immediate battle is usually not appropriate, and a long-term strategy considering future battles is required. That is the most crucial factor in turn-based RPG.

Battles in turn-based RPG proceed based on turns. In each turn, each character performs one action in order, which is typically decided by a parameter called speed. Fig. 2 shows an example of a sequence in one turn in a two-versus-two battle, including characters' status information. In the figure, the square boxes next to the characters show the characters' names, health points (HP)/maximum HP, magic points (MP)/maximum MP, attack (ATK), defense (DEF), speeds (SPD), and possible actions, from the top and the left.

Non-battle events occur outside battles, such as inns where players can recover their characters, blacksmiths who reinforce players' weapons, item shops providing items for the next expedition, and treasure chests where players can get weapons during the expedition. Settings of the non-battle events can also significantly affect the difficulties or strategies of the battles. Therefore, it is not advisable to design battle and non-battle events separately.

Players in turn-based RPGs advance the stories by repeating non-battle events and battle events while determining their choices in each event. In this sense, players may prefer stages requiring them to make various decisions (e.g., focusing on weak enemies first or trying to defeat all encountered enemies). Conversely, if all decisions from players produce similar results, they may get bored with the game.

B. Desirable Stages

Turn-based RPGs usually contain several separated scenes, such as the base towns, the dungeons in which enemies appear, and the fields connecting scenes to others. Fig. 3 shows an example of a simplified turn-based RPG. At the base town, characters can buy items and heal their wounds. However, in order to recover damages received in dungeons or fields, characters need to consume medicine, use magic, or find a recovery point. The game flow can be roughly grouped into four courses in general. The four courses, as shown in Fig. 3, are (a) leveling up and collecting items, (b) exploring the world, (c) going through dungeons and defeating mid-bosses, and (d) moving to other base towns. Players play the game by repeating the four courses as they like, with the goal of defeating the final boss.

Different designs are required for different courses. In this study, we skip the designs of the stories, the videos, and the audio of turn-based RPGs and target the relations between events for course (c). Usually, when players want to defeat a mid-boss in a dungeon, their characters need to explore the whole dungeon from the beginning to the boss without withdrawing or being defeated. Otherwise, they need to start from the very beginning of the dungeon next time. Defeating enemies in the dungeon helps the player characters to level up so that they may become strong enough to defeat the mid-boss. However, in order to start the mid-boss battle with a good condition, they also need to manage the medicine or magic available to recover damage properly in intermediate battles. The following are factors that we consider to make players feel satisfied with the course (c) stages.

- Which strategies are valid is a crucial issue in RPGs. We consider that stages are not enjoyable when no strategy is effective or all of them are. Also, for stages with different designs and settings, it is boring if players can stick to a single strategy, or some specific strategies are obviously too good or bad. Stages should be designed so that players can enjoy thinking about their choices.

TABLE I
COMPARISON OF TYPICAL TURN-BASED RPGS AND OUR PLATFORM

	Typical turn-based RPG	Research platform
Stage structure	diverse stage structures	one-way battle and recovery events
Game objective	defeating the boss after a long journey	defeating the boss after proceeding events
Number of team members	≥0	0
Character parameters	many parameters	HP, ATK, SPD
Order of actions in battles	by SPD with unique rules	player character's SPD > enemies'
Actions in battles	various skills	attack, retreat
Reward of winning a battle	leveling up, gold, items, etc	increasing ATK (10%)
Result of retreating a battle	game over, nothing happen, etc	losing HP (15%)

TABLE II
COMPARISON BETWEEN THREE KINDS OF PCG METHODS

	training data	evaluation function	learning cost	generation cost
PCGML	necessary	–	high	low
Search-based PCG (GA)	–	necessary	–	high
Reinforcement Learning	–	necessary	high	low

- Enemies should have proper strength according to the timing of their appearance. Hard to defeat strong enemies in early stages or too-weak enemies in late stages are unreasonable.

Game balancing has been known to be crucial but difficult, and turn-based RPGs are such an example [46]. Even in simpler designs (e.g., our platform described in the next section), satisfying the factors mentioned above is not easy, not even to say more complex designs, e.g., skill-based systems that players can try all available skills in the skill-spaces without designer-imposed limitations [47].

C. Research Platform

As mentioned in the previous section, we only focus on the relations between events in this study, particularly battle events and recovery in non-battle events. As there was no proper environment for our research, we designed an extensible platform that contains the most fundamental elements in turn-based RPGs. Table I summarizes the comparison between typical turn-based RPGs and our platform. Depending on the settings, our platform can be adapted to various types of turn-based RPGs. Section IV-B formulates the stage generation problem in our platform into a general MDP problem. We implemented a simplified version for the experiments.

IV. APPROACH

In this section, we first describe general ideas for applying RL to PCG in Section IV-A. Next, we present the MDP formulation of our platform in Section IV-B, which is required when applying RL for stage generation. The MDP formulation includes state and action representations, state transition function, and reward function. For the reward function, we design an evaluation function to rate completed stages. Section IV-C introduces a method called *virtual simulation* to help improve the quality of generated stages, where we give virtual simulated rewards to non-rewarded actions during training. Moreover, Section IV-D introduces two methods for increasing the diversity of generated stages. One is *randomized event initialization*, which starts to generate stages from randomly initialized events. The other is *diversity-aware greedy policy*, which tries to generate diverse stages while not degrading the quality of the stages. Finally, a brief comparison of Khalifa et al.'s work [20] and ours is made in Section IV-E.

A. Reinforcement Learning Applied to PCG

As a novel AI-based approach for PCG, we proposed to apply reinforcement learning (RL) [21][22]. Table II compares PCGML, search-based PCG, and RL from different aspects. For example, PCGML requires training data, while the other two require evaluation functions to tell how promising the generated content looks. Both PCGML and RL need a relatively long learning time. However, once the learning is finished, the generation costs are low. In contrast, search-based PCG does not require a learning process but consumes time when generating content, which is more challenging for online generation. To sum up, RL's advantages include low generation costs and no need to prepare training data.

To apply RL, we need to formulate PCG problems as MDP. The following explains how MDP elements (i.e., state, action, transition probability function, and reward function) may look like using an example shown in Fig. 4, which tries to generate levels consisting of three sequential events. At the initial state, the first event has been determined as ev_x while the rest two remain blank. An action is to add an event to the next blank one. For simplicity of discussions, let the state transition be deterministic. Namely, when adding an event, say ev_y, the next blank event always becomes that one. Thus, states in this PCG problem are levels with different event combinations, including blank events. After all three events are decided, we can use an evaluation function to get the quality of the complete level, which can be used as the reward function in the MDP.

We believe that the ideas of formulating PCG into MDP and applying RL are general and can handle many game genres. As long as the MDP formulation of a PCG problem is finished, RL algorithms can be applied to learn good policy (i.e., what actions to take for given states) based on the provided reward function. After the learning, the policy serves as the content generator at relatively low generation costs.

B. MDP Formulation for Our Turn-Based RPG Stages

In this section, we explain how to formulate stage generation in our turn-based RPG into MDP.

1) Stage Representation: Each stage in our platform is represented by a real number matrix, indicating a series of one-way battle events and recovery events, with a boss event at the end. An empty stage is represented by a zero matrix filled with

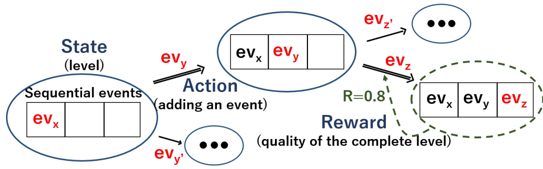

Fig. 4. An illustration of the MDP for generating game content using levels consisting of three sequential events as an example.

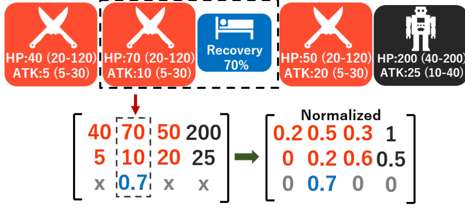

Fig. 5. Example of converting the battle-battle-recovery-battle-boss stage to a 3×4 matrix.

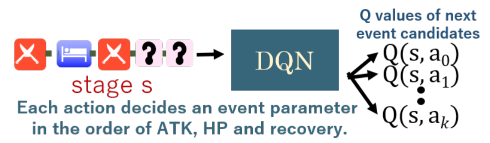

Fig. 6. DQN for stage generation in this paper.

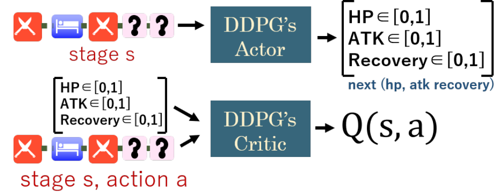

Fig. 7. DDPG for stage generation in this paper.

zeros. In the matrices, each column contains three values, i.e., enemy's HP, enemy's ATK, and player character's recovery. In more detail, the former two form a battle event, and the last one forms a recovery event. Fig. 5 is a simple example consisting of five events: enemy-enemy-recovery-enemy-boss.

We represent enemies' HP and ATK values by integers in some reasonable ranges. In Fig. 5, the second enemy's HP is 70, while the range is 20 to 120. Similarly, the ATK is 10, while the range is 5 to 30. For bosses, both lower bounds and upper bounds for HP and ATK are usually higher than those of enemies. To make our representation generalize to any ranges, we normalize the values into $[0, 1]$ according to the given ranges. For example, HP of 70 in the range of 20 to 120 is normalized to (70-20)/(120-20)=0.5.

As for the third element in a column, the recovery event, the value ranging from 0 to 1 represents the player character's recovery rate relative to max HP. For example, assuming that the player character's HP/max HP is 20/100, a recovery rate of 0.7 makes the player character's HP become 20+0.7×100=90. The max HP bounds the recovery, i.e., even with a recovery rate of 1.0, the player character's HP becomes 100 instead of 120 after the recovery event.

It can be seen that even in the same range of $[0, 1]$, the values mean quite different things between battle events and recovery events. Also, battle events require two values, while recovery events only require one. Considering the topological structure of stage representation, we attach one recovery event after every battle event. However, it is unusual that each battle event follows a recovery event in turn-based RPG stages. For battles without recovery events (the 1st and 3rd columns of the matrix in Fig. 5), the recovery values are set to 0. In this way, a series of battle-recovery events, including battles without immediate recovery events and the final boss, can be represented by concatenating columns with three real number values ranged in $[0, 1]$. A stage containing b battle events is represented by a $3 \times (b+1)$ matrix, the *+1* for boss.

2) Action Representation: Actions in our platform are defined to fill in the stage matrices sequentially. In other words, an action decides one or some real number values of the given incomplete stage matrix. Many classical RL algorithms, including deep Q-network (DQN) [43], assume discrete actions, i.e., action spaces being finite sets. Meanwhile, algorithms coping with continuous actions, such as deep deterministic policy gradient (DDPG) [48], are also proposed. We define both discrete and continuous action representations so that our platform is applicable to both kinds of RL algorithms.

For discrete actions, we divide the range of $[0, 1]$ into k sections, and thus, there are $k+1$ actions. Each action stands for the lower bound of the corresponding section, except that the $(k+1)$st action stands for the overall upper bound, i.e., 1. Assuming $k = 100$, the 101 actions are 0, 0.01, 0.02, 0.03, ..., and 1. DQN takes an incomplete stage matrix as the input and outputs $k+1$ action values (Q-values), as shown in Fig. 6. The $k+1$ Q-values represent how the $k+1$ possibilities of the next parameter look promising for the given stage. The interpretation of *the next* may vary in different implementation. In this paper, we fix the order to 1st enemy's HP $\rightarrow$ 1st enemy's ATK $\rightarrow$ 1st recovery rate $\rightarrow$ ... $\rightarrow$ the boss's ATK. In this manner, one stage parameter is determined at a time by selecting the one with the highest Q-value or by methods that introduce exploration such as ϵ-greedy.

Considering that the three values in the same column may influence each other, it is more natural to output them at a time. However, the case of discrete actions falls into the curse of dimensionality. In the example, the output increases to $101^3 = 1030301$, an extremely large value that may cause problems during learning. Continuous action algorithms directly output numerical values in the given ranges. Thus, it is easy to output three values at once, representing one column. This is done by the *actor* of DDPG, while the Q-value is evaluated by another model called the *critic*, as shown in Fig. 7. More specifically, the actor takes an incomplete stage matrix as the input and outputs three real number values. For example, (0.5, 0, 0.8) means a 50% of enemy's HP in the given range, a minimum value of enemy's ATK, and a recovery rate of 0.8. The critic then takes the same stage matrix and the actor's action as the input and outputs the Q-value.

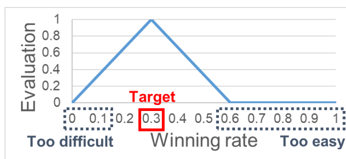

Fig. 8. An evaluation based on a winning rate. (x-axis: winning rate, y-axis: evaluation based on the winning rate)

3) State Transition: Given an incomplete stage matrix as a state and a real number value (or a column) as an action, the matrix is always filled in by the given value(s) exactly. For example, given a zero matrix and a value of 0.5, the first element in the first column of the matrix becomes 0.5, meaning that the enemy in the first battle event is assigned an HP of 50% in the given range. Stage generation terminates when the matrices are fully filled.

4) Reward Function: In this work, our purpose is to create stage generators that can generate enjoyable stages. We define a stage evaluation function and use it as the reward function of the MDP. The evaluation function focuses on player engagement from seven perspectives, as explained in Appendix A, where the major one is the appropriateness of difficulty. Although there are various possibilities to define the quality of stages, the difficulty is considered one of the most important factors that influences player engagement [49] [50] [51]. Also, the difficulty assessment is different from game to game. As a simple example, we define a strategy to be a sequence of action selections from the beginning to the end of a stage and then use the ratio of winning strategies among all possible strategies as the difficulty of our turn-based RPG stages. Generally speaking, good stages should have moderate ratios of winning strategies to avoid situations like players cannot win whatever they do or can win whatever they do.

In more detail, players have two available actions, attack and retreat, for each battle event in our platform. When there are $b + 1$ battle events including a boss event, the possible number of strategies is 2^b as it is not allowed to retreat in the boss battle. In this condition, it is evident and deterministic whether a boss is beatable or not. We calculate the ratio of defeating the boss among all strategies as the winning rate to represent the difficulty of a stage.

Deciding the proper target winning rate on our game setting could be an issue. However, the preferred difficulty level varies depending on the games, and it is difficult to say which values are the most reasonable, which we leave out of the scope of this paper. In this study, we assume 30% to be the most favorable winning rate and leave it as a tunable parameter[2]. In our evaluation function, as shown in Fig. 8, the scores are 0 when the winning rates are higher than 60% and linearly increase from 0 to 1 in the range of 0%-60% as the winning rates go closer to 30%.

[2]Some readers may consider that a winning rate of 30% results in too difficult stages. However, our definition of the winning rate implies that the player selects each action with equal probabilities (i.e., a random player). Rational players usually do not play randomly, which we expect to have higher chances to clear the stages.

There are many other entertaining factors in addition to difficulty, and those factors are different for each player [52]. Accurately predicting the satisfaction of human players is not the subject of this paper, so we only consider some common factors, such as dramatic surviving and tough wins. The details of the evaluation function are explained in Appendix A. Although this specific evaluation function is applied in our method, it can be replaced by any other calculable evaluation functions. Similar to search-based PCG, adaptive generation can be achieved by adjusting the evaluation function.

C. Method for Improving Stage Quality

With the MDP in the previous section, we can apply RL to learn to generate stages; however, applying simple RL algorithms is insufficient to get stages with high quality, i.e., stages that make appropriate balances between battle and recovery events. One problem is that rewards can be obtained only after whole stages are generated under our evaluation function. In other words, immediate rewards are all 0 except for the last actions that complete stages. This is an instance of the temporal credit assignment problem [53], a long-studied subfield of RL. Conceptually speaking, the problem aims to figure out how each action among a sequence of actions influences the final outcome. When the reward signal is sparse or delayed, RL may get stuck in local optima, especially for deep neural networks.

To address the temporal credit assignment problem, various methods were proposed, which can be roughly divided into two groups [54]: assigning correct credit based on gradients and based on extra values or targets. As an example of the former, Ferret et al. [55] introduced a new transfer learning approach that used a self-attentive architecture to assign credit in a backward view. For the latter, the main idea is to use surrogate rewards for the actions. For example, Arjona-Medina et al. [56] and Liu et al. [57] decomposed the returns of episodes back to the actions. Harutyunyan et al. [58] assigned credits according to the likelihood that the actions led to the observed outcome. Yu et al. [59] applied a Monte-Carlo method to estimate the rewards of intermediate actions in the work of SeqGAN, which aimed to generate sequences of discrete tokens (e.g., sentences).

In this work, we propose virtual simulation (VS) to provide intermediate actions with rewards by a Monte-Carlo method, where the ideas are similar to SeqGAN. For each intermediate action, the algorithm applies the current policy with an exploration policy, such as ϵ-greedy, to generate several completed stages. The average evaluation of these stages then serves as the immediate reward of the action. The experimental details and settings are in Section V-A1. By doing this, we do not need extra efforts to design immediate rewards for intermediate actions. The method is expected to be more time-consuming for learning, but the offline cost is not a significant problem in PCG. One of the major benefits of RL is that even if learning takes a long time, it is possible to provide content immediately once the learning is over, i.e., online generation.

D. Methods for Increasing Stage Diversity

One primary goal of PCG is to automatically (or semi-automatically) provide players with diverse content even for the same game. For example, after clearing a stage, players may want to try different settings other than the same one, or they may get bored soon. It is better that the method can generate several diverse and good stages instead of only the best one. However, this goal is not considered in the proposed RL as long as the diversity of stages is not included in the reward function. Our designed evaluation function indicates how good one given stage is. To consider diversity, several stages should be compared at once, which is expected to be complicated. Instead of revising the evaluation function, we propose two ways to address the diversity issue.

1) Randomized Event Initialization: We propose to assign random parameters to several beginning events for each stage instead of an empty stage as the first method to address diversity[3]. In other words, the zero matrices are assigned with some random values as first elements and then used as the initial states of stage generation. We consider this method to be able to balance the quality and the diversity of stages to some extent in the following senses. For an extreme end, assume that no events are randomly initialized and a model has been well trained by RL. When the model is used to generate stages (for players to play), it is common to select the best actions (i.e., event parameters). As a result, only one good stage can be generated. For the other extreme end, when all event parameters are randomly assigned, the stages will have a high diversity but are highly likely to have low quality.

A reasonable idea in between is to randomly initialize several events and make the RL model finish the rest. During training, RL algorithms are also provided with such randomly initialized stages as the initial states. We expect that RL algorithms can learn how to generate good stages from random initialization. In this way, with diverse initial stages, we can generate diverse and good stages. However, in some cases, even if the initial stages are different, the later parts may end up being similar, especially when long stages are generated. Lack of diversity can be alleviated by randomly initializing more events, while this may result in stages with low evaluations. For this method, the number of events to initialize determines the trade-off between the quality and diversity of stages, and detailed results will be shown in Section V-C.

2) Diversity-Aware Greedy Policy: Getting diversity from the initial stages is not enough as the diversity only depends on the initial states. As the second method to address diversity, we propose to sample *not-bad-but-distant* actions based on Q-values, which we call diversity-aware greedy policy (DAGP)[4]. Compared to the greedy policy that always takes the best actions, the proposed method may take worse actions where stochastic noise is introduced for increasing stage diversity. In this way, even when the generation starts with the same initial stage, we can still get quite different stages.

The most critical part of DAGP lies in how to select not-bad-but-distant actions. If improper noise is introduced, stages with low evaluations are likely to be generated. The method has two prerequisites: (a) good actions distant from the best-evaluated ones exist, and (b) the Q-values of those good actions can be estimated with some accuracy. The experiments in Section V-B will confirm that the two prerequisites are indeed satisfied. From the results, we conclude that it is possible to generate diverse and high-quality stages based on Q-values and present a noise introduction algorithm to select such not-bad-but-distant actions as follows.

i) Decide the stage structure and train the model as usual.
ii) Determine where and how noise is introduced (noise event). The place can be fixed in advance or randomly selected during generation. Also, determine the number of candidate stages n and a threshold d for event parameters' distance to the best. In this paper, the distance between a given set of event parameters and another set is defined by the Euclidean distance.
iii) Except for the noise events, greedily decide actions (event parameters) according to the learned policy. In the case of noise events, generate n random actions as candidates. Reject those whose distances to the actions by the greedy policy are less than d, as it means that they are close to the best action. Among the remaining candidates, select the one with the highest Q-value.

E. Comparison with Khalifa et al.'s Work

Khalifa et al. [20] also applied RL to PCG to generate game levels and formulated the problem into MDP. The two terms, stage and level, can be used interchangeably to indicate structural units of games. In the comparison, theirs are referred to as levels following their paper and ours as stages. Their levels were represented by 2D integer matrices, indicating the layouts of the maps where different integers stand for different objects. The layouts were initialized randomly, and the actions were to *change* the objects of tiles in the maps. We also represent stages by matrices while the meaning is different, which are the settings of battle and recovery events. In both designs, the actions modify stages (levels), where their approach changes existing values and ours assigns new values. They proposed three ways to locate the value to change, and one of which following a predefined order is similar to ours.

Their reward functions were manually designed to reflect whether the levels got closer to the goals of the game. For example, since PacMan has only one player, adding a player object when there is none receives positive rewards. Their goal was to generate playable levels that obey the game rules, while ours focuses on the balance between events. They required an additional function to determine whether the goals were reached so that the generation process could be terminated.

To achieve the diversity of levels, they set a parameter called change percentage, which limited the numbers of tiles that could be changed from the initial levels. Smaller change percentages were suggested since too-high values were expected to cause the generator to override most of the initial levels aiming at few optimal solutions. We also set random initial values, though our approaches differ in terms of theirs. In addition, we introduce noise to choose not-bad-but-distant actions to make the generated stages more diverse.

[3]The method was called *random initial stage* in our previous work [22].
[4]The method was called *stochastic noise policy* in our previous work [22].

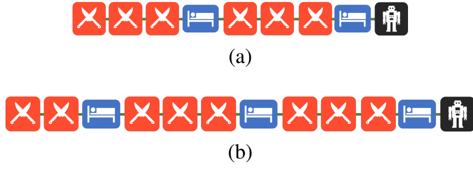

Fig. 9. Composition of the stage having (a) nine and (b) twelve events, where each square represents battle (red), recovery (blue), and boss (black) events.

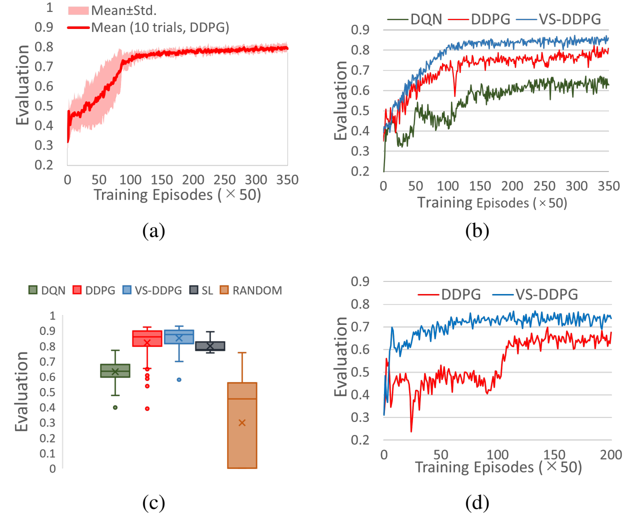

Fig. 10. The average evaluations of 50 stages: (a) DDPG's 10-trial training curve of the 9-event stage, (b) DQN, DDPG, VS-DDPG's training curves of the 9-event stage, converging around 0.65, 0.83, 0.87, (c) box plot of the 9-event stage after training, and (d) DDPG and VS-DDPG's training curves of the 12-event stage, converging around 0.65 and 0.75. For training curves, the action selection involved exploration, such as ϵ-greedy.

V. Experiments and Discussions

In this section, the results of generating stages with high quality are included in Section V-A. The learned Q-values are analyzed in Section V-B. Finally, the results of generating diverse stages are presented in Section V-C.

A. High-Quality Stage Generation

We employed two RL algorithms with different action spaces to generate turn-based RPG stages by the proposed PCG approach. More specifically, the two RL algorithms were DQN [43] for discrete actions and DDPG [48] for continuous actions. We also included two non-RL methods for comparison, though different kinds of methods have different assumptions and advantages, as discussed in Section IV-A, and might not be directly comparable. The random generation method decided event parameters by a uniform distribution within $[0, 1]$. The supervised learning (SL) method took an incomplete stage as the input and decided the parameters in the next column (i.e., enemy's HP, enemy's ATK, player's recovery rate), similar to the actor of DDPG.

As described in Section IV-B1, we represented turn-based RPG stages by matrices of three rows and $b{+}1$ columns, which contain b battle events and a boss battle. For the third row, if there was no recovery event after the battle, the value was fixed to 0. The number of battles b and the stage structure (i.e., the arrangement of battle and recovery events) were determined in advance and then fixed.

1) Experimental Setup: We conducted the experiments under the following settings.

- The execution environment and the network settings are described in Appendix B.
- The composition of a stage was the one from Fig. 9a, and a longer stage (Fig. 9b) was used to compare the effect of virtual simulation. Each stage was represented by $3{\times}7$ and $3{\times}9$ matrices, respectively. Among these, the event parameters in the first one or two columns were determined randomly, which were initial states.
- As described in Section IV-B2, the significant difference between DQN and DDPG is the action space. DQN was for discrete actions, and DDPG for continuous actions. In our setting, DQN had 101 actions[5], each representing a possible setting (0, 0.01, ..., 1) for the next event parameter (enemy HP, enemy ATK, or player's recovery rate). DDPG's actor output three real values in $[0, 1]$ at once for the event parameters in the same column. The designs led to a minor difference in how many times of input/output were required to complete a stage. For the example of Fig. 9a, assume that one column of the stage matrix is initialized. DQN requires $3{\times}6$ because it decides event parameters sequentially one at a time (even when there is no recovery event). In contrast, DDPG only requires 6 because it decides one column at a time.
- Virtual simulation presented in Section IV-C was applied to DDPG (abbr. VS-DDPG). The number of virtual simulating episodes was 5, and virtual simulation was applied from the beginning of the training.
- SL had the same network structure as the actor of DDPG, except that batch normalization layers were added to prevent overfitting. In practice, preparing training data with high quality is a critical issue for SL methods. In this experiment, we simply employed the random generation method and only collected stages whose evaluations were over 0.7. It cost 10 hours to obtain 20000 nine-event stages (Fig. 9a). For more complex games, we expected the preparation of training data to be more costly.

2) Result: Since randomness was involved in the training, we investigated how training curves were influenced and ran DDPG under the same setting for 10 trials. DDPG was trained for $50{\times}350{=}17500$ episodes to generate 9-event stages. Fig. 10a shows the mean evaluations and the standard deviations of the 10 trials. Each data point was the average evaluation from 50 episodes, where one episode means generating one stage from an initial one. Note that the evaluations were collected from stages generated during training, which means that the exploration was also involved. The standard deviations before $50{\times}100$ episodes were relatively high, mainly because of higher exploration (ϵ-greedy and OU-noise) in the early

[5]When doing discretization, proper settings of granularity (i.e., number of actions) depend greatly on the ranges of event parameters. In some preliminary experiments, we obtained poor results if the number of actions was too low (e.g., 11). Considering that discretization is a general and challenging issue, detailed investigations are made out of this paper, which we focus more on that the proposed approach is applicable to both discrete and continuous actions.

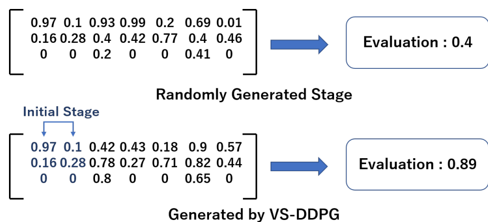

Fig. 11. Stage examples (top: random generation, bottom: VS-DDPG), where each column represents event parameters of battle and recovery, i.e., HP, ATK, and recovery rate from the top.

phases of training. In addition, we tried several network structures and hyperparameters, obtaining similar results. In the following experiments, we only show the results of one trial under the settings described in Section V-A1.

Fig. 10b shows the training curves of DQN, DDPG, and VS-DDPG on generating 9-event stages. The results showed that both DDPG and VS-DDPG were better than DQN (under our experiment settings). Especially, VS-DDPG converged at the highest evaluation value of 0.87. Fig. 10c shows the box plot of evaluation values of 50 stages after training, including two baselines, random generation and SL. Stages made by the random generation had a wide range of evaluation values, and most stages were lowly evaluted. As for the remaining four, (VS-)DDPG generated stages with the highest average evaluations, though some bad stages were also obtained. SL could generate highly evaluated stages most consistently, but the average evaluations were slightly worse than (VS-)DDPG. Assume that we only want to collect stages whose evaluations were over 0.9. Taking DDPG and the random generation as examples, the former could obtain one such stage every 3-4 trials (about 0.2 seconds), while the latter required about 2000 trials (80 seconds). DDPG's speed was about 400 times faster than random generation.

We further conducted an experiment to compare DDPG and VS-DDPG for generating longer stages. The results of the 12-event stages (Fig. 9b) are shown in Fig. 10d. As expected, VS-DDPG performed obviously better than DDPG as the problem of delayed rewards was considered more serious for longer stages. About the training time of 9- and 12-event stages for 10000 episodes, VS-DDPG took 300 and 2500 minutes, while DDPG took 60 and 160 minutes. Even under the same training time, DDPG could not reach the same evaluation values.

Fig. 11 shows two stages generated by random (top) and VS-DDPG (bottom). The upper stage has strong enemies in the early phases with low recovery and somewhat weak enemies in the later phases; hence, it has a lower evaluation value. In contrast, the bottom one is generated from the first two columns of the upper one, but it has a better evaluation value by filling adequate values with implicitly considering the requested conditions by the reward function.

As described above, if an evaluation function is given, the RL model can learn a policy that can generate highly evaluated stages. We have shown that our proposed approach for generating high-quality stages is promising.

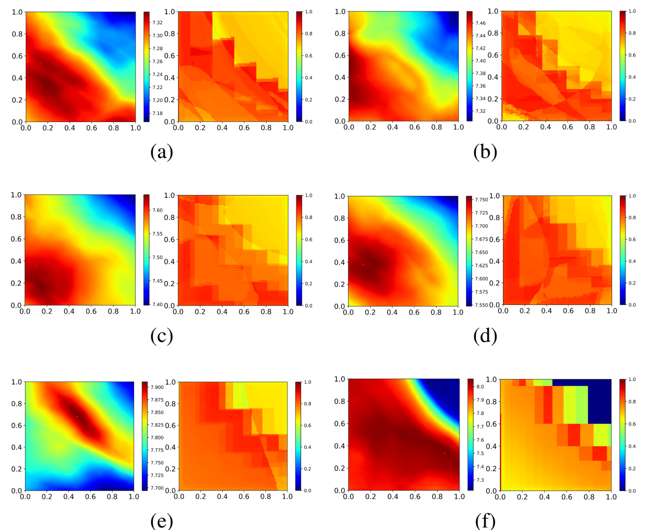

Fig. 12. Distribution of Q-values (a-f left) and evaluation values (a-f right) for the 2nd-7th battle events (x-axis: HP, y-axis: ATK). Colors closer to red have higher values.

B. The Relation Between Q-value and Evaluation

In the previous section, generating good stages is the main goal. In this section, we further shift the goal to generate diverse stages while maintaining quality. The policy of RL generally takes the best-evaluated actions, but when considering diversity, it is better to choose other actions. For actions other than the best-evaluated ones, two questions about evaluations arise: (1) whether a distant action from the best-evaluated one gets an extremely worse evaluation value and (2) whether the Q-values of distant actions are inaccurate due to lack of learning. Therefore, we first compare the distributions of Q-values and evaluation values to confirm the distribution of good actions and how trustworthy Q-values are.

1) Experimental Setup: From Section V-A, VS-DDPG was clearly better than others and was thus used in the rest of the experiments. The stage composition was the same as Fig. 9a, and the first one battle event was randomly initialized. The actor of VS-DDPG then completed the stage. The Q-value distribution for each battle event was collected from the critic.

Unlike the critic of VS-DDPG, which can also evaluate incompleted stages, our evaluation function requires completed stages. Thus, we applied the actor of VS-DDPG to finish the rest of the events by selecting the most promising actions and then used the evaluations of such completed stages for incomplete stages. The evaluations were the highest ones the actor could reach.

2) Distribution of Q-values and Evaluation Values: For VS-DDPG, we obtained the Q-value distribution by sampling actions and inputting them into the critic. An action was represented by a three-dimensional vector of $[0, 1]$ (HP, ATK, recovery rate). To make the results easier to understand, we used the recovery rate from the actor and varied HP and ATK.

Fig. 12a (left) shows the distribution of the second battle event's Q-values, with the first battle event randomly initialized. The Q-values are the highest around (0.1, 0.55). When both go too high, which means the enemy is too strong in early

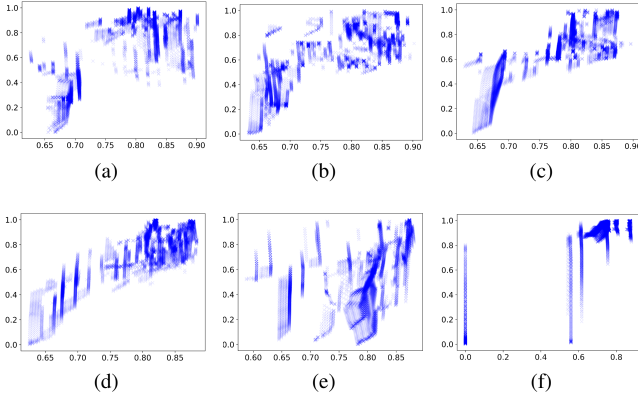

Fig. 13. Relations between Q-values and evaluation values from Fig. 12 for the 2nd-7th battle events, where the Pearson correlation coefficients are 0.742, 0.787, 0.806, 0.849, 0.383, and 0.884. (x-axis: evaluation values, y-axis: normalized Q-values)

stages, the Q-values become lower. The scale of the Q-values is 7.18 to 7.32, which shows the difference gap is small, and the value is overestimated than the evaluation value. Lillicrap et al. [48] have pointed out such overestimations of Q-value when the target problem is complicated.

Fig. 12a (right) shows the distribution of the evaluation values of the second battle event. It can be seen that multiple good actions exist. Also, the general trends between Figs. 12a left and right are similar. The distributions of the Q-values and evaluation values for the 3rd-7th battle events are drawn in Fig. 12b-12f in similar ways. The results also demonstrate that the critic of VS-DDPG learns the general trends.

Fig. 13 shows the relations between the Q-values normalized to $[0, 1]$ and the evaluation values from Fig. 12. The Pearson correlation coefficients were 0.742, 0.787, 0.806, 0.849, 0.383, and 0.884, respectively. Except for Fig. 13e, all had highly positive correlations. In other words, the Q-values by the critic were generally trustworthy.

C. Diverse Stage Generation

In this section, we first define two indicators to evaluate the diversity of the generated stages and then demonstrate the effectiveness of our approaches proposed in Section IV-D.

1) Diversity Assessment: The first indicator calculates the average squared difference (ASD) between the event parameters of two stages. Higher values of parameter ASD mean that two stages have distant HP, ATK, and recovery rates.

The other is the number of winning strategies that are different in two stages. As introduced in Section IV-B4, winning strategies are those that can beat the boss. In the experiments, the stages contained six enemy events and a boss event, i.e., Fig. 9a. For each enemy event, two actions, attack and retreat, were available. Thus, the total number of possible strategies was 2^6. A higher difference means that efficient strategies in one stage do not work in the other, and thus players need to try different actions for different stages.

2) Result: Our approaches for diverse stage generation require a well-trained model. In the experiments, we employed VS-DDPG for generating stages and evaluating actions. The stage composition was the one in Fig. 9a. During training, we randomly selected an integer $1 \leq c \leq 5$ and initialized the first c columns to random values, where it was $1 \leq c \leq 2$ in the previous experiments. This modification aimed to explore more stages and learn Q-values better.

TABLE III
AVERAGE ASDS AND AVERAGE REWARDS FROM 200 STAGES (FIG. 9A) GENERATED WHEN THE FIRST c COLUMNS WERE RANDOMLY INITIALIZED.

c	1	2	3
ASD	0.727 (±0.024)	1.040 (±0.030)	1.290 (±0.020)
Reward	0.797 (±0.006)	0.766 (±0.006)	0.711 (±0.015)
c	4	5	6
ASD	1.443 (±0.028)	1.552 (±0.027)	1.742 (±0.027)
Reward	0.667 (±0.013)	0.598 (±0.006)	0.533 (±0.018)

We first experimented on randomized event initialization (REI) and generated 200 stages for each integer $c \in [1, 6]$ where the first c columns were randomly initialized. Table III shows the average ASDs and rewards for each c. Note that the stages contained seven columns, so randomly initializing six meant that almost all event parameters were randomly decided. As expected, with more randomly-initialized columns, the average ASDs increased while the average rewards decreased.

As discussed in Section IV-D and shown in Table III, the diversity was limited by only employing REI, especially when high quality was also desired. In the next experiment, our proposed DAGP (diversity-aware greedy policy) was applied, where the first column was randomly initialized, and the rest were set to noise events. We prepared 50 initial stages to see whether different initial stages influence DAGP's results. We tried several values for d (distance threshold) and n (candidate stage number), and each setting generated 50 stages from each initial stage. The goal was to investigate how different stages could be generated from the same initial one. High values of d prevented actions from being too close to the actor's action, and thus, we expected to generate diverse stages. Also, we expected high values of n (many candidate actions) to increase the chances of obtaining good stages.

Fig. 14 shows the average reward, the parameter ASD, and the number of different winning strategies of different d and n settings. Note that when calculating the parameter ASD and the number of different winning strategies, a stage was compared only to another that was generated from the same initial stage, instead of calculating among all 50×50 stages. The results in Fig. 14 were the averages over the 50 initial stages. As expected, higher n indeed led to stages with higher average rewards, as shown in Fig. 14a, and higher d led to higher parameter ASD and the number of different winning strategies, as shown in Figs. 14b and 14c. The only exception for diversity was the number of different winning strategies when d was 0.7. The reason was related to the low quality of stages, either impossible to clear or too easy to clear.

Except the case of $d = 0.7$ in Fig. 14c, we observed that the quality (average reward) had an opposite trend to the diversity (parameter ASD and number of different winning strategies). We suspected that the phenomenon was related to the Q-values learned by the critic, as shown in Fig. 12. Even though multiple actions had high evaluation values (Fig. 12a-12f (right)), the

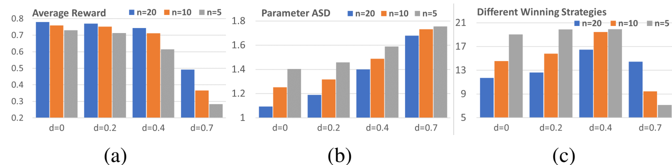

Fig. 14. Evaluation of DAGP results averaged from 50 initial stages where each had 50 stages generated. (a) average reward, (b) parameter ASD, (c) number of different winning strategies. Each blue, orange, and gray bar is for n = 20, 10, and 5, respectively, and each group of bars represents the result when d = 0, 0.2, 0.4, 0.7 from the left.

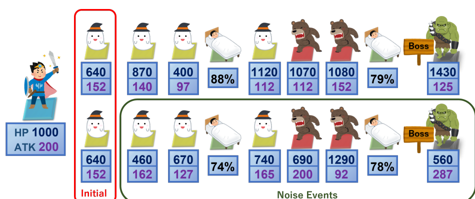

Fig. 15. Stages generated by the actor policy (top) and DAGP with the first event fixed and the rest set to noise events (bottom).

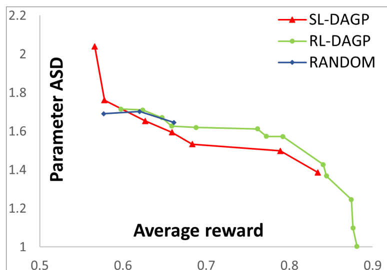

Fig. 16. The Pareto frontier to average rewards and parameter ASD for RL-DAGP, SL-DAGP, and random generation.

critic tended to have Q-values centered at one of the actions (Fig. 12a-12f (left)). Since DAGP selected actions with the highest Q-values from far-enough actions, when d decreased or when n increased, it was more likely to get an action closer to the actor's action. Thus, increasing n was likely to decrease the diversity, while decreasing d was likely to increase the average rewards.

Fig. 15 shows example stages by VS-DDPG (top) and DAGP (bottom). The former was evaluated as 0.889, and the later as 0.878. The stage parameter ASD between the two stages was 1.449, and the number of different winning strategies was 11. Although the first event was the same, it could be seen that different good stages were generated by introducing noise.

Although DAGP was shown to be able to generate diverse stages, one minor problem remained. Sometimes, DAPG generated stages with low evaluations (e.g., 0.3), which should be discarded to keep high quality. We further conducted an experiment to investigate the influence on quality and diversity by filtering out bad stages. Since such removal was expected to increase the cost of online generation, we also investigated the generation cost. In addition to DAGP with VS-DDPG (abbr. RL-DAGP), we included two other algorithms for comparison, DAGP with SL (SL-DAGP) and random generation.

To apply DAGP in SL, we added a network similar to the critic of DDPG that evaluated given stage-action pairs. The evaluator network was trained by 10000 randomly generated stages regardless of their evaluations. Different from the generator network that only required good stages, the evaluator network's training data should also contain bad stages so that it could predict the evaluations of state-action pairs better. For the three algorithms, we prepared five initial stages and let each setting collect 50 stages from each initial stage. DAGP parameters were $n \in \{5, 10, 20\}$ and $d \in \{0, 0.2, 0.4, 0.7\}$, and stages with evaluation values lower than v were discarded, $v \in \{0, 0.5, 0.7, 0.8, 0.85\}$. Note that a v of 0 meant that no stages are discarded. We omitted the results of settings that could not obtain 50 stages within 500 trials since those settings would require more than 3 seconds to get one stage, which we considered unsuitable for online generation.

Fig. 16 shows the Pareto frontier to the average rewards and parameter ASD. RL-DAGP dominated SL-DAGP in most cases. Random generation could generate diverse stages but the quality was much lower. As discussed in Section V-A2 (the 2nd paragraph), when the random generation was employed to collect good stages with evaluations over 0.9, it required about 80 seconds to get one stage, which was time-consuming. From the experiments, RL-DAGP was the most promising to generate high-quality and diverse stages online.

VI. Conclusion and Future Work

In this paper, we proposed the use of reinforcement learning with the theme of stage generation in turn-based RPG. The stage generation problem was formulated into a Markov decision process, where states were incompleted or completed stages represented by real-number matrices and actions were to fill in the matrices. We defined an evaluation function to consider the difficulty and several entertaining factors of stages. By using the evaluation function to give rewards in reinforcement learning, our proposed method successfully generated highly evaluated stages.

We applied DQN and DDPG for discrete and continuous actions, respectively. For discrete actions, the range was divided into a fixed number of sections, and one value in the stage matrix was decided at a time. In contrast, it was easier for continuous actions to decide several values (of related events) at once, which was expected to handle related events more effectively. DDPG indeed generated better stages than DQN under our experiment settings. When our proposed method is used in other PCG problems, action types may be a factor that affects the performance.

Under our reward design, the delayed rewards made the model hard to efficiently learn to evaluate actions, especially when the event sequence was long. To overcome the difficulty, we introduced virtual simulations to provide intermediate actions with rewards. Experiments showed that virtual simulations helped to improve the quality of stages.

Since the primary purpose of this research is to generate good and diverse stages, we proposed randomized event initialization that assigns random parameters to several beginning events and a diversity-aware greedy policy that chooses actions

distant but not bad compared to the best actions from the model's view. The experiments demonstrated the effectiveness of the proposed approaches in obtaining good and diverse stages for online generation.

Future research includes improving each method and showing that it can work in larger, more diverse, and more complex games. For example, for 2D video games, our design is expected to be more easily adapted to generate levels in platformer games (e.g., Mario) in the sense that we can design the content from left to right. For levels where tiles' relations in other directions are important (e.g., Sokoban), some additional efforts are needed. Another possible research direction is to design evaluation functions that can reflect human players' feelings in playing games and then verify the effectiveness. We expect that our method is able to try to maximize any given evaluation functions as the current one is already considerably complicated. For example, we can try to conduct subject experiments for collecting human players' evaluations and using supervised learning to approximate those evaluations. By using the supervised learning model as the evaluation function, the method should be able to generate stages that fit the players' preference.

TABLE IV
USED TOOLS AND VERSION

Tool	Spec (version)		
gpu	NVDIA GeForce GTX 1070		
cpu	Intel Core i7-7700 3.60GHz		
ram	16GB	tensorflow-gpu	1.10.0
cuda and cudnn	8.0 and 6.0.21	keras	2.2.2

TABLE V
DQN AND DDPG SETUP

Parameter	Value	
	DQN	DDPG & VS-DDPG
layers	Conv 36→64→ FC 128→256→256→256	Conv 256→256→256→ FC 128→128→128
memory size	300000	
learning rate	0.25×10^{-4}	Actor: 1×10^{-5} Critic: 1×10^{-4}
target network	hard update (3000)	soft update (0.001)
batch size	128	64
discount factor	0.9	
exploration	ϵ-greedy $(1 \to 0.1)$	OU noise with ϵ-greedy $(1 \to 0.1)$

APPENDIX A
EVALUATION FUNCTION

The employed evaluation function is weighted from seven sub-functions, $f(x) = \Sigma_{i=1}^{7} w_i f_i(x)$, with considering the following features of stage x: The number of events (n_{events}), the number of winning strategies (n_{win}), the number of the strategies that retreat from the ith enemy ($n_{retreat_i}$), the number of strategies that have dramatic surviving moments ($n_{dramatic}$), the number of winning strategies that have monotonous actions (n_{mono}), character recovery rate at the ith recovery event, ($c_{recover_i}$), the ith enemy's normalized ATK and HP within [0, 1] (e_{ATK_i} and e_{HP_i}).

- f_1: the winning rate soundness. The winning rate should not be too low nor too high, as shown in Fig. 8, $w_1 = 0.4$.
- f_2: bonus for dramatic surviving. Players often feel pleasant when surviving dramatically from a crisis (when reaching the ith recovery event with an HP $\leq 0.3\times$(max HP) and $c_{recover_i} \geq 0.5$). $n_{dramatic}/n_{win}$, $w_2 = 0.1$.
- f_3: bonus for moderate parameter ranges. Enemies or recovery events within moderate ranges (enemy parameters within [0.05, 0.95] and recovery rates within [0.2, 0.8]) seem more natural. $(\Sigma_i g(c_{recover_i}, 0.2) + \Sigma_i \min(g(e_{ATK_i}, 0.05), g(e_{HP_i}, 0.05)))/n_{event}$, where $g(x, y) = \min(x/y, (1-x)/y, 1)$, $w_3 = 0.2$.
- f_4: bonus for tough wins. If the character after defeating the boss has a low HP ($\leq 0.4\times$(max HP)), it means that the stage is challenging. Average of $\min((1 -$ final HP(%)$)/0.6, 1)$ in winning strategies, $w_4 = 0.1$.
- f_5: penalty on early escapes. If the strategy of escaping in the early battles are effective, the game flow may be considered unnatural. $1 - (3n_{retreat_1} + 2n_{retreat_2} + n_{retreat_3})/6n_{win}$, $w_5 = 0.05$.
- f_6: penalty on weak enemies in later phases. It is irrational that later enemies are too weak. $1 - (0.7 - \min(0.7, (\Sigma_{i \in \{\text{last two enemies}\}}(e_{ATK_i} + e_{HP_i}))/4))/0.7$, $w_6 = 0.1$.
- f_7: penalty on monotonous strategies. Winning a stage with monotonous strategies, i.e., only attacks or only retreats, can make players boring. $1 - 0.5 \times n_{mono}$, $w_7 = 0.05$.

APPENDIX B
NETWORK SETUP

The codes were implemented in Python 3.6, and the used libraries and the machines for experiments are listed in Table IV. The network settings of DQN, DDPG, and VS-DDPG are listed in Table V, where Conv x means a convolutional layer with x filters of size 1×3 and FC x means a fully-connected layer with x nodes.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Numbers JP18H03347, JP17K00506, JP20K12121.

REFERENCES

[1] A. Summerville and M. Mateas, "Super mario as a string: Platformer level generation via lstms." in Proc. 1st Int. Joint Conf. DiGRA/FDG, 2016.
[2] J. Dorsey and H. Rushmeier, "Advanced material appearance modeling," in *ACM SIGGRAPH*, 2009.
[3] A. Liapis, C. Holmgård, G. N. Yannakakis, and J. Togelius, "Procedural personas as critics for dungeon generation," in *Applications of Evolutionary Computation*, 2015, pp. 331–343.
[4] D. Gravina and D. Loiacono, "Procedural weapons generation for unreal tournament iii," in *2015 IEEE Games Entertainment Media Conference (GEM)*, 2015, pp. 1–8.
[5] G. Pickett, F. Khosmood, and A. Fowler, "Automated generation of conversational non player characters," in *INT/SBG@AIIDE*, 2015.
[6] N. Justesen, R. Rodriguez Torrado, P. Bontrager, A. Khalifa, J. Togelius, and S. Risi, "Illuminating generalization in deep reinforcement learning through procedural level generation," *NeurIPS Workshop on Deep Reinforcement Learning*, 2018.

[7] Nintendo, *Kyoto, Japan*, Super Mario Bros., 1985.
[8] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith, and S. Risi, "Evolving mario levels in the latent space of a deep convolutional generative adversarial network," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2018, p. 221–228.
[9] J. Togelius, R. De Nardi, and S. M. Lucas, "Towards automatic personalised content creation for racing games," in *2007 IEEE Symposium on Computational Intelligence and Games*, 2007, pp. 252–259.
[10] D. Loiacono, L. Cardamone, and P. L. Lanzi, "Automatic track generation for high-end racing games using evolutionary computation," *IEEE Transactions on Computational Intelligence and AI in Games*, pp. 245–259, 2011.
[11] Red Hook Studios, *Vancouver, Canada*, Darkest Dungeon, 2016.
[12] Chunsoft, *Tokyo, Japan*, Mystery Dungeon series, 1993.
[13] Nippon Ichi Software, *Gifu, Japan*, Disgaea, 2003.
[14] A. Summerville *et al.*, "Procedural content generation via machine learning (pcgml)," *IEEE Transactions on Games*, vol. 10, no. 3, pp. 257–270, 2018.
[15] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations, ICLR*, 2014.
[16] van den Oord et al., "Conditional image generation with pixelcnn decoders," in *Advances in Neural Information Processing Systems 29*, 2016, pp. 4790–4798.
[17] I. Goodfellow *et al.*, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., 2014, pp. 2672–2680.
[18] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation," in *Applications of Evolutionary Computation*, 2010, pp. 141–150.
[19] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural content generation*. Springer, 2016.
[20] A. Khalifa, P. Bontrager, S. Earle, and J. Togelius, "Pcgrl: Procedural content generation via reinforcement learning," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, no. 1, pp. 95–101, Oct. 2020.
[21] S. Nam and K. Ikeda, "Automatic generation of states in turn-based rpg using reinforcement learning," in *2018 Game Programming Workshop (GPW)*, vol. 2018, nov 2018, pp. 160–167.
[22] S. Nam and K. Ikeda, "Generation of diverse stages in turn-based role-playing game using reinforcement learning," in *2019 IEEE Conference on Games (CoG)*, 2019, pp. 1–8.
[23] D. Gravina, A. Khalifa, A. Liapis, J. Togelius, and G. N. Yannakakis, "Procedural content generation through quality diversity," in *2019 IEEE Conference on Games (CoG)*, Aug 2019, pp. 1–8.
[24] A. Liapis, G. N. Yannakakis, and J. Togelius, "Enhancements to constrained novelty search: Two-population novelty search for generating game content," in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, 2013, p. 343–350.
[25] D. Hooshyar, M. Yousefi, M. Wang, and H. Lim, "A data-driven procedural-content-generation approach for educational games," *Journal of Computer Assisted Learning*, vol. 34, no. 6, pp. 731–739, 2018.
[26] A. Zook, S. Lee-Urban, M. R. Drinkwater, and M. O. Riedl, "Skill-based mission generation: A data-driven temporal player modeling approach," in *Proceedings of the The Third Workshop on PCG'12*, p. 1–8.
[27] Y. Liang, W. Li, and K. Ikeda, "Procedural content generation of rhythm games using deep learning methods," in *Entertainment Computing and Serious Games*, 2019, pp. 134–145.
[28] M. Kaidan, C. Y. Chu, T. Harada, and R. Thawonmas, "Procedural generation of angry birds levels that adapt to the player's skills using genetic algorithm," in *IEEE 4th Global Conference on Consumer Electronics*, 2015, pp. 535–536.
[29] T. Oikawa, C.-H. Hsueh, and K. Ikeda, "Improving human players' t-spin skills in tetris with procedural problem generation," *16th Advances in Computer Games (ACG 2019)*, 2019.
[30] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on Computational Intelligence and AI in Games*, 2011.
[31] M. Stephenson and J. Renz, "Procedural generation of complex stable structures for angry birds levels," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 2016, pp. 1–8.
[32] A. Alvarez, S. Dahlskog, J. Font, and J. Togelius, "Empowering quality diversity in dungeon design with interactive constrained map-elites," in *2019 IEEE Conference on Games (CoG)*, 2019, pp. 1–8.
[33] S. Lee, A. Isaksen, C. Holmgård, and J. Togelius, "Predicting resource locations in game maps using deep convolutional neural networks," in *Proc. Artif. Intell. Interactive Digit. Entertainment Conf*, 2016.
[34] A. Summerville, M. Behrooz, M. Mateas, and A. Jhala, "The learning of zelda: Data-driven learning of level topology," *in Proc. 10th Int. Conf. Found. Digit. Games*, 2015.
[35] M. Awiszus, F. Schubert, and B. Rosenhahn, "Toad-gan: Coherent style level generation from a single example," *Artificial Intelligence and Interactive Digital Entertainment. AAAI*, pp. 10–16, 2020.
[36] R. Rodriguez Torrado, A. Khalifa, M. Cerny Green, N. Justesen, S. Risi, and J. Togelius, "Bootstrapping conditional gans for video game level generation," in *IEEE Conference on Games (CoG)*, 2020, pp. 41–48.
[37] E. Giacomello, P. L. Lanzi, and D. Loiacono, "Doom level generation using generative adversarial networks," in *2018 IEEE Games, Entertainment, Media Conference (GEM)*, 2018, pp. 316–323.
[38] K. Park et al., "Generating educational game levels with multistep deep convolutional generative adversarial networks," in *2019 IEEE Conference on Games (CoG)*, 2019, pp. 1–8.
[39] R. Jain, A. Isaksen, C. Holmgard, and J. Togelius, "Autoencoders for level generation, repair, and recognition." In ICCC Workshop on Computational Creativity and Games, 2016.
[40] A. Sarkar, Z. Yang, and S. Cooper, "Controllable level blending between games using variational autoencoders," in *Proceedings of the EXAG Workshop at AIIDE*, 2019.
[41] S. Thakkar, C. Cao, L. Wang, T. J. Choi, and J. Togelius, "Autoencoder and evolutionary algorithm for level generation in lode runner," in *2019 IEEE Conference on Games (CoG)*, 2019, pp. 1–4.
[42] A. Sarkar, A. Summerville, S. Snodgrass, G. Bentley, and J. Osborn, "Exploring level blending across platformers via paths and affordances," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, no. 1, pp. 280–286, 2020.
[43] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, feb 2015.
[44] M. Guzdial, N. Liao, and M. Riedl, "Co-creative level design via machine learning," in *Proceedings of the EXAG Workshop at AIIDE*, vol. 2282, 2018.
[45] E. C. Sheffield and M. D. Shah, "Dungeon digger: Apprenticeship learning for procedural dungeon building agents," ser. CHI PLAY '18 Extended Abstracts, 2018, p. 603–610.
[46] W. J. Kavanagh, A. Miller, G. Norman, and O. Andrei, "Balancing turn-based games with chained strategy generation," *IEEE Transactions on Games*, pp. 1–1, 2019.
[47] A. Pantaleev, "In search of patterns: Disrupting rpg classes through procedural content generation," ser. PCG'12. Association for Computing Machinery, 2012, p. 1–5.
[48] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, 2016, Conference Track Proceedings*.
[49] O. Missura and T. Gärtner, "Player modeling for intelligent difficulty adjustment," in *Discovery Science*, 2009, pp. 197–211.
[50] M. Lankes and A. Stoeckl, "Gazing at pac-man: Lessons learned from a eye-tracking study focusing on game difficulty," in *ACM Symposium on Eye Tracking Research and Applications*, 2020.
[51] B. Bostan and S. Ogut, "Game challenges and difficulty levels: lessons learned from rpgs," in *International Simulation and Gaming Association Conference*, 2009.
[52] N. Sato, K. Ikeda, and T. Wada, "Estimation of player's preference for cooperative rpgs using multi-strategy monte-carlo method," in *2015 IEEE Conference on Computational Intelligence and Games*, pp. 51–59.
[53] M. Minsky, "Steps toward artificial intelligence," *Proceedings of the IRE*, vol. 49, no. 1, pp. 8–30, 1961.
[54] A. P. Badia *et al.*, "Agent57: Outperforming the Atari human benchmark," in *Proceedings of the 37th International Conference on Machine Learning*, vol. 119. PMLR, 2020, pp. 507–517.
[55] J. Ferret, R. Marinier, M. Geist, and O. Pietquin, "Self-attentional credit assignment for transfer in reinforcement learning," in *IJCAI*, 2020, pp. 2655–2661.
[56] J. A. Arjona-Medina, M. Gillhofer, M. Widrich, T. Unterthiner, J. Brandstetter, and S. Hochreiter, "Rudder: Return decomposition for delayed rewards," in *Advances in Neural Information Processing Systems*, 2019.
[57] Y. Liu, Y. Luo, Y. Zhong, X. Chen, Q. Liu, and J. Peng, "Sequence modeling of temporal credit assignment for episodic reinforcement learning," CoRR, vol. abs/1905.13420, 2019.
[58] A. Harutyunyan *et al.*, "Hindsight credit assignment," in *Advances in Neural Information Processing Systems*, 2019, pp. 12 488–12 497.
[59] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient," in *Proceedings of the Thirty-First AAAI'17 Conference on Artificial Intelligence*, p. 2852–2858.