# Using Reinforcement Learning to Generate Levels of Super Mario Bros. with Quality and Diversity

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

***Abstract*—Procedural content generation (PCG) is essential in game development, automating content creation to meet various criteria such as playability, diversity, and quality. This paper leverages reinforcement learning (RL) for PCG to generate Super Mario Bros levels. We formulate the problem into a Markov decision process (MDP), with rewards defined using player enjoyment-based evaluation functions. Challenges in level representation and difficulty assessment are addressed by conditional generative adversarial networks (CGAN) and human-like AI agents that mimic aspects of human input inaccuracies. This ensures that the generated levels are appropriately challenging from human perspectives. Furthermore, we enhance content quality through virtual simulation (VS), which assigns rewards to intermediate actions to address a credit assignment problem (CAP). We also ensure diversity through a diversity-aware greedy policy (DAGP), which chooses not-bad-but-distant actions based on Q-values. These processes ensure the production of diverse and high-quality Super Mario levels. Human subject evaluations revealed that levels generated from our approach exhibit natural connection, appropriate difficulty, non-monotony, and diversity, highlighting the effectiveness of our proposed methods. The novelty of our work lies in the innovative solutions we propose to address challenges encountered in employing PCGRL in Super Mario Bros, contributing to the field of PCG for game development.**

***Index Terms*—Reinforcement learning, procedural content generation, Super Mario Bros., quality and diversity.**

## I. Introduction

PROCEDURAL generation is a versatile technique widely employed across various applications, offering advantages in cost-effectiveness and quality assurance. One of its most prominent domains is game development. Procedural content generation (PCG) employs algorithms to create diverse in-game content autonomously, ensuring replayability and entertainment value while managing financial and memory constraints. Its capabilities have been demonstrated in various aspects of games, including generation of levels [1] [2] [3], characters [4], and items [5]. Given its vital role in game development, extensive research efforts have been dedicated to the development and application of PCG, spanning various game genres such as platformers [6], rhythm games [7], and puzzle games [8].

This paper proposes a PCG method aiming to generate levels in Super Mario Bros. In Super Mario, a level consists of tile arrangements, with each level breaking down into patterns. It is crucial to generate content that aligns with the standards of game developers, addressing aspects such as difficulty, entertainment, or playability. These qualification indicators should be based on human players' perceptions to ensure a satisfactory and engaging gaming experience. In Super Mario levels, the layout, types of tiles, and proper pattern placement are vital for enjoyable levels. Improper concatenation can result in levels that are unnatural, overly complex, repetitive, or non-playable. Therefore, these factors are often considered as part of the quality criteria for Super Mario levels. Diversity of the generated content is also essential since players expect new content with every launch. In addition to quality criteria and diversity, the appropriate difficulty is also crucial to the gameplay experience from a human perspective. Many existing works employ human-like AI agents to evaluate the level of difficulty, representing different aspects of human likeness. Some studies employed biological constraints, such as confusion [9] and physical fatigue from successive inputting keys [10] to impart human-like behavior to AI. In our work, we focus on AI agents that mimic new aspects of human input inaccuracies time span. This approach ensures that the generated levels have appropriate difficulty for human players.

Two primary PCG techniques widely adopted in existing studies are procedural content generation via machine learning (PCGML) [11] and search-based PCG [12]. Each has its advantages and disadvantages, making them suitable for different applications and addressing various practical constraints in game development scenarios. PCGML relies on substantial training data and tends to produce less diverse content when faced with limited datasets. It proves useful for games with abundant and easily obtainable human-generated content. On the other hand, search-based PCG does not require extensive training but relies on evaluation functions to tailor content to meet designer or player preferences. The genetic algorithm (GA) is often employed to optimize the evaluation values. However, GA may require modifications and parameter setups to generate diverse content [13]. Additionally, optimization processes can be time-consuming and may not be suitable for immediate content generation needs. Real-world game development often encounters challenges of insufficient training data and the requirement for immediate content, deeming PCGML and search-based PCG impractical for such cases. To address these challenges, many studies employ the reinforcement learning (RL) method in PCG, so-called PCGRL, to enable immediate game content generation without requiring substantial data.

This study employs the PCG using the RL method proposed in our previous works [14] and [15] to generate high-quality

The authors are with Sirindhorn International Institute of Technology, Thammasat University, and Japan Advanced Institute of Science and Technology, e-mail: (sanggyu@siit.tu.ac.th; hsuehch@jaist.ac.jp; pavinee.rerk@dome.tu.ac.th; kokolo@jaist.ac.jp)

and diverse Super Mario levels. The general idea of formulating the level generation problem into a Markov decision process (MDP) is similar to that presented in [15]. However, differences in formulating each problem into MDP and the characteristics of the games pose two distinct challenges. This paper aims to address them to ensure that the generated levels meet our defined quality and diversity criteria.

Firstly, due to the complexities in the Super Mario game, the action dimension of RL is larger. We adopted twin delayed deep deterministic policy gradient (TD3), and its training was proven efficient in environments with less than 10 dimensions of action space (e.g., Hopper-v1 with 3 dimensions or Walker2d-v1 with 6 dimensions) [16]. However, the training becomes difficult for environments with more than 10 dimensions (e.g., Humanoid-v1 with 17 dimensions) [17]. Therefore, we decided to use low-dimensional vectors as the output of RL and map them to the corresponding level pattern.

The second challenge is the evaluation of the difficulty of generated content. In Super Mario, the difficulty is also an essential factor, especially based on human perception. In our previous work, we evaluated the difficulty based on the characters' every possible action. The action choices in Super Mario are broader and deeper than those in turn-based RPGs, which makes it very difficult to apply the previous approaches. In this work, we adopt human-like AI agents and assume they represent human players. By letting the AI agents play the levels, the degree of difficulty from human ability is measured based on their play logs.

Notably, both our research and that of Shu et al. [18] employ PCGRL to generate Mario levels. Both studies represent the generated levels as latent vectors (matrices) and assess their playability using the A* agent. However, our research introduces an additional layer of evaluation by employing human-like AI agents that emulate human behavior, allowing us to assess not only playability but also the difficulty of the levels, ensuring that the generated content is both playable and appropriately challenging. We also ensure natural connectivity among adjacent patterns using conditional generative adversarial networks (CGAN), which is introduced in our previous work [14].

In terms of addressing level monotony, Shu et al.'s research primarily compares the occurrences of individual tiles, while our work shifts the focus to the arrangement of patterns. Our approach ensures that levels do not have the same adjacent patterns, as we do not want users to experience similar patterns consecutively. This can directly impact playstyle and is a critical aspect of both monotony and the overall entertainment value of the levels.

Regarding generating diverse levels, Shu et al. utilized historical deviation to encourage the RL agent to create novel patterns. In contrast, we achieve diversity through randomized initialization (RI) and the implementation of a diversity-aware greedy policy (DAGP). DAGP ensures diversity by generating patterns different from those produced by RL's greedy policy, using distance criteria without compromising quality. This approach enables us to generate multiple high-quality levels that exhibit diversity from one another.

Lastly, Shu et al. employed a CNet-assisted evolutionary repairer [19] to address noise in the generation process, while our approach utilizes pattern matching to find the most similar patterns from existing data.

In summary, our paper employs a PCG technique using RL to generate Super Mario levels. The significance of this work lies in its innovative solutions to various challenges encountered in PCGRL and its application to Super Mario content generation, contributing to the advancement of both game development and procedural content generation research. Our work builds upon these foundations and introduces several key contributions to the field as follows:

- We validate our framework of applying RL to the PCG problem proposed in [15], using more complex target content—Super Mario levels. Several challenges arise due to the different characteristics of the games, and we proposed innovative solutions to address them.
- We define different quality criteria for Super Mario levels, encompassing difficulty based on humans' perspectives, non-monotonous layout, and playability. By generating levels that meet these criteria, we ensure high-quality gameplay experiences. In addition, the diversity among generated levels is assured by introducing RI and DAGP without compromising their quality.
- We evaluate the difficulty of our levels using AI agents that mimic human nature in terms of inaccuracies in input timing and input time span, providing a more realistic assessment of gameplay challenges.

The remainder of this paper is organized as follows: In Section II, we discuss the quality criteria in Super Mario levels and provide an overview of existing PCG techniques. Section III describes the gameplay and components of Super Mario levels. Section IV presents the level generation processes. Section V presents the concept of our proposed PCGRL considering quality and diversity criteria. Section VI outlines the experimental results and discussions. Finally, Section VII concludes the paper, discussing limitations and outlining future works.

## II. Background

PCG is a technique that employs algorithms to generate diverse game content automatically. This generation can occur either upon players' requests (online) or during the game development stage (offline). Due to time constraints, online generation is more challenging. PCG aims to produce high-quality content that ensures an enjoyable gameplay experience, whether for online or offline generation. Additionally, content diversity plays a pivotal role in enhancing the long-term appeal of games, providing players with fresh and engaging experiences in each gameplay.

### A. Quality Criteria

Various aspects of content quality, corresponding to players' enjoyment, have been explored in previous studies. These aspects include difficulty [20], [21], [22], [23] and entertainment value [24]. A review paper by Gravina et al. [25] also emphasizes the need for PCG techniques to maximize both quality and diversity as such aspects are essential to expanding

the capabilities of PCG in games, ensuring enjoyable and engaging gaming experiences.

In the assessment of quality criteria for Super Mario levels, numerous studies have proposed different aspects such as the shape of levels [26], difficulty, or playability [27]. For instance, Shi and Chen [28] introduced playability, resource balance, difficulty curve, reachability, and aesthetic properties as qualitative labeling criteria for levels. Level difficulty, defined by configurations of enemy and hole tiles, is dynamically adjusted based on AI agents' performance indicators. In Flimmel et al. [29]'s work, the quality of levels is evaluated through non-linearity, difficulty, and complexity. Non-linearity aims to prevent entirely flat levels, while difficulty is estimated as the total difficulty value of each enemy/hole tile type, assessed based on the authors' sentiments. Complexity ensures that levels present a variety of obstacles. Zhang et al. [30] proposed a PCG method that repairs broken Super Mario levels to meet aesthetics and playability requirements. Beukman et al. [31] used diversity and difficulty as their evaluation metrics. Diversity among the two levels is compared by trajectories of actions obtained from A* AI agent, and difficulty is measured by the degree of explorations A* AI agent makes throughout a level. Thus far, the criteria for quality and diversity in Super Mario levels require further exploration, considering different aspects and combinations of criteria to enhance gameplay engagement and entertainment levels. In this paper, we define quality criteria for Super Mario levels, encompassing difficulty based on human perspectives, non-monotonous layout, and playability.

While many existing PCG techniques exist, two representative techniques rooted in the artificial intelligence concept have gained widespread recognition in the existing literature: search-based PCG and PCGML. The details of each technique are introduced in the following sections.

### B. Search-Based PCG

Search-based PCG is a generate-and-test algorithm known for its efficiency in generating high-quality content, even for complex games, as discussed in a survey paper by Togelius et al. [32].

Search-based PCG primarily applies GAs to optimize the evaluation function value. GAs are well-known evolutionary algorithms that involve iterative operations resembling the natural selection process, continually improving the generated content compared to previous iterations. Furthermore, search-based PCG enables developers to customize evaluation functions based on specific criteria tailored to developers' or players' preferences. For example, Togelius et al. [33] applied search-based PCG to generate racing tracks in racing games, using evaluation functions that maximize the entertainment value of the track, accounting for variations in specific players' driving styles.

Despite its efficiency, search-based PCG has significant content diversity and generation time drawbacks. Due to the diversity issue of GAs, content generated using search-based PCG may exhibit redundancy. To address this, efforts have been made to incorporate diversity-related aspects into the generation process. For instance, Loiacono et al. [34] introduced elements such as track curvature, speed profile, and surrounding landscapes to enhance diversity in racing game tracks. Similarly, Alvarez et al. [35] utilized a variation GA called MAP-Elites algorithm to ensure diversity when generating designer-interactive dungeons.

Additionally, regardless of quality and diversity requirements, search-based PCG often involves numerous evaluation processes (trials and errors), which can be time-consuming. Consequently, this technique may not be well-suited for online content generation.

### C. PCG via Machine Learning (PCGML)

PCGML is a framework for generating content using various machine learning algorithms. Unlike search-based PCG, PCGML directly generates the content from the trained model using the existing content as training data. Many studies demonstrated the use of PCGML to generate various kinds of game content.

For example, Summerville and Mateas [36] employed long short-term memory (LSTM) networks, known for their sequence prediction capabilities, to generate Super Mario levels. These levels were represented as strings, with each character denoting a tile on the map. By providing an initial seed sequence (a series of pre-assigned tiles), the LSTM generated complete levels tile by tile. Lee et al. [37] applied the convolutional neural network (CNN) to predict resource locations within StarCraft II maps, showcasing the potential value of PCGML for map designers. Similarly, Summerville et al. [38] used Bayesian networks to learn the room-to-room structures of Zelda dungeons, enabling the generation of new levels with statistical properties similar to the given examples.

In PCGML, GAN and variational autoencoder (VAE) are widely utilized for their ability to generate content based on input datasets. GAN, for instance, has been successfully applied to PCG for generating levels in Super Mario [39], Zelda [40], and Doom [41]. Similarly, VAEs have been useful in generating levels for Super Mario [42], [43], Lode Runner [44], and various other platformer games [45]. An example of GAN usage in PCG is found in the work of Abraham and Stephenson [46], where they employed GANs to generate several structures composed of small blocks in Angry Birds using an encoding/decoding process. These structures are similar to patterns in this work.

While PCGML holds significant potential, challenges arise in its application when confronted with a shortage of adequate training data. Determining which features to use in content generation and the type of data to collect can also be complicated. For instance, in the case of Super Mario levels, it may be challenging to ensure that collected level data aligns with appropriate difficulty levels or specific play styles. This underscores the importance of understanding the content's characteristics during data collection, which can be costly and intricate. Noise is likely to be included. Additionally, content structure varies between games, making it challenging to reuse content from one game to generate content in another.

Despite these challenges, various strategies have been employed to address these limitations of PCGML. Siper et al.

[47], [48] implemented Path of Destruction techniques, locally modifying data to replicate additional training data from existing sources. Schubert et al. [49] adopted a tokenization approach on each tile, leveraging downsampling techniques to generate tokens and utilizing TOAD-GAN for level generation from a limited dataset. Torrado et al. [40] utilized a bootstrapping technique to augment the playable levels in training data. Bontrager and Togelius [50] introduced a Generative Playing Network comprising an agent and a generator. The agent plays given levels to enhance its proficiency, while the generator aims to create playable levels. Despite these efforts, the impact of data quantity persists in the outputs, particularly concerning quality and diversity, especially within complex game domains. In such scenarios, the application of PCGRL can be promising.

### D. Reinforcement Learning

In real-world game development, several challenges, such as limited access to training data and the need for online content generation, often hinder the practicality of procedural content generation techniques like search-based PCG and PCGML. To address these issues, RL has emerged as a promising approach. RL is a mathematical framework widely used in scenarios where an agent interacts with an environment to achieve specific goals while receiving feedback in the form of rewards, either positive or negative.

RL typically adopts the concept of the MDP, represented as $(S, A, P, R, \gamma)$, where $S$ denotes the state space. $A$ represents the action space. $P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the probability of transitioning from state $s$ to state $s'$ by taking action $a$ at time $t$. $R_a(s, s')$ is the immediate reward resulting from the transition. $\gamma \in [0, 1]$ is the discount factor, determining the importance of future cumulative rewards compared to immediate rewards.

One significant challenge in RL is the credit assignment problem (CAP), where immediate rewards may not clearly reflect the environment's ultimate goal. Frequently, rewards for actions are sparse or delayed until the end of an episode, making it difficult to understand how each action in a sequence impacts the final outcome. To address CAP, we adopt Yu et al.'s concept in SeqGAN [51], in which the Monte Carlo method is employed to estimate intermediate action rewards aiming to generate sequences of discrete tokens in sentence generation.

RL distinguishes itself by its ability to self-learn through interactions with the environment, eliminating the need for training data. It can guarantee the optimal policy when state-action pairs and their associated Q-values are known. However, achieving this ideal condition is a formidable task in most real-world applications due to the vast number of state-action pairs. To address this complexity, we use function approximators, such as CNN, to estimate Q-values, enabling RL to navigate complex state-action spaces effectively.

As shown in Table I, RL stands out among other PCG techniques, particularly in scenarios where online generation and training data are crucial. PCG techniques have been extensively employed to generate game levels in the literature. For instance, Khalifa et al. [52] have demonstrated the feasibility of PCGRL for generating game levels on simplified research platforms. To formulate the generation problem into MDP, they introduced three representation modules: Narrow, Turtle, and Wide. Additionally, they employed the Change Percentage to finish the episode of RL. Earle et al. [53], and Jiang et al. [54] employed PCGRL that provided users with a degree of control over maze generation outcomes. In their work, Earle et al. trained the model using conditional input and reward shaping by encoding the goal into the state representation of the level, while Jiang et al. prepared multiple generators and evolved generators based on criteria. Delarosa et al. [55] introduced a co-creation design tool utilizing a mixed-initiative approach, balancing user input and AI suggestions through RL. Zakaria et al. [56] proposed a controllable generator capable of producing playable levels without the need for tailoring the reward function. Instead of adopting conditional input and reward shaping, they utilized the denser feedback from multiple level sizes, starting from small to desirable size.

TABLE I
COMPARISON BETWEEN THE THREE 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 |

## III. SUPER MARIO LEVELS

This paper builds upon our previous research, where we successfully applied our PCG method to a relatively simple turn-based RPG. In this study, we extend our approach to the iconic platformer game Super Mario to showcase its adaptability and effectiveness in more complex gaming environments. This section offers insights into Super Mario levels as the target content. It discusses the characteristics of desirable levels and the configuration of the two Super Mario platforms employed.

### A. Components

Super Mario is a world-famous platformer game developed by Nintendo. During the gameplay, players control Mario, passing through various obstacles and reaching the goal on the right-most side within the time limit. A Super Mario level is characterized by a two-dimensional layout featuring various types of tiles, which serve as either obstacles or useful items. In this work, we divide a level into a set of fixed-size tile patterns as shown in Fig. 1.

Obstacles within these levels encompass enemies, holes, and walls, while useful items include coins and mushrooms. Certain tile types, enemy behaviors, and complex elements have been excluded from our research. Table II summarizes the 6 tile types (total 14 tiles) utilized in this study. In the gameplay, Mario cannot pass through wall types except destroyed brick tiles, which can be demolished through the head-bump action while powered up. Mario or any objects that

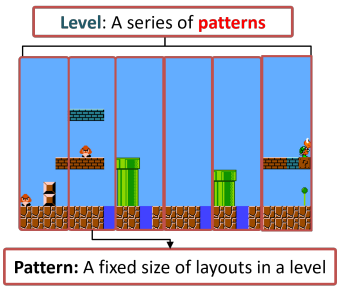



Fig. 1. An example of tile patterns in Super Mario levels

TABLE II
MARIO TILES USED IN THIS PAPER

| Tile | Tile Type | Tile | Tile Type | Tile | Tile Type | Tile | Tile Type |
|---|---|---|---|---|---|---|---|
| (brick) | wall | (coin brick) | wall | (coin) | item | (coin box) | wall |
| (item box) | wall | (empty block) | wall | (block) | wall | (block) | ground |
| (pipe) | wall | (hole) | hole | (goomba) | enemy | (koopa) | enemy |
| (shell) | enemy | (goal) | etc | | | | |

fall into a hole are destroyed. Enemy tiles can cause damage to Mario upon physical contact. The arrangement of wall, hole, and enemy tiles significantly influences gameplay enjoyment and difficulty, as levels with excessive wall stacks, wide holes, or an abundance of enemies can become overly challenging and unplayable.

### B. Desirable Characteristics

Super Mario has a diverse player base with varying skill levels, making appropriate and diverse difficulty levels essential for an enjoyable gaming experience [57], [58], [59]. Additionally, avoiding monotony is crucial in level design [60]. Monotony occurs when levels feature redundant patterns that require repetitive actions, potentially leading to player disengagement. In this paper, we define desirable level characteristics in Super Mario as follows:

- Appropriate difficulty and playability: Levels should be playable and match players' perceptions of challenge. This involves well-placed enemy and hole tiles, significantly impacting a level's difficulty. The level's challenge can be assessed by analyzing human players' play logs.
- Non-Monotonous gameplay: Engaging gameplay relies on diverse action sequences. When similar patterns are placed locally, players may repeatedly perform the same actions in a short time. Thus, level design should prioritize the concatenation of patterns with various action sequences, ensuring a richer and more engaging gaming experience.
- Natual connection: A well-made level comprises subsequent patterns with similar features in appearance to the previous ones. Levels become unnatural when different features of patterns are concatenated abruptly.

This study adopts cloned versions of Super Mario. Some features are excluded because they exist in a few specific levels or are too complicated. Two clone platforms, Python Mario [61] and Mario AI Framework [62], are employed.

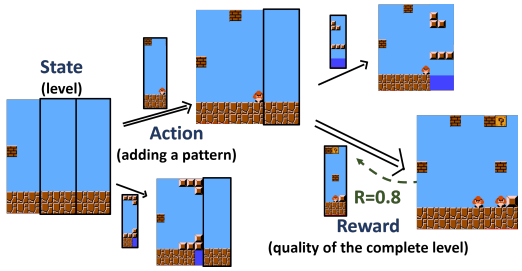



Fig. 2. Illustrations of MDP for generating levels

## IV. Generation of Super Mario Levels

RL typically employs the MDP framework for problem formulation. In our case, we use MDP to represent the generation of Super Mario levels, as illustrated in Fig. 2. Each action corresponds to adding a pattern to a blank slot, with deterministic state transitions resulting in the exact pattern from the action being added. This process continues until all blank slots are filled, at which point the evaluation function is used to provide a reward.

This MDP-based approach is versatile and adaptable across various game genres. By formulating the problem as an MDP, RL algorithms can learn effective policies (i.e., which actions to take given certain states) based on provided reward functions. Once the learning process is complete, this policy can efficiently serve as a content generator, offering substantial cost-effectiveness benefits.

Our approach to Super Mario level generation faces two key challenges associated with the dimensionality of RL states and actions and the evaluation of difficulty as elaborated below:

*1) Dimension of RL States and Actions:* A Super Mario level has a 15×n two-dimensional structure. The level generation process progressively adds 15×4 tile patterns to the level matrix, starting from the left-most side. Given this setup, it is logical to represent Super Mario levels as states of MDP. Consequently, the 15×4 patterns serve as the actions for the RL agent.

However, training RL models with high-dimensional continuous action spaces can be challenging, as discussed in [17]. To address this challenge, we employ low-dimensional vectors as the output of the RL agent and subsequently map these vectors to their corresponding level patterns. This transformation from vectors to patterns and vice versa also presents some difficulties. Therefore, we utilize CGAN as a function approximator to output the appropriate pattern based on the vector input. The detailed descriptions of the MDP formulation methodology and function approximator are explained separately in Sections IV-A and IV-C.

*2) Difficulty Evaluation:* Assessing the difficulty of generated Super Mario levels is essential to ensure an engaging and appropriately challenging gaming experience. One common method is to evaluate the difficulty of using play logs, which can be costly when involving human resources. To overcome this limitation, we introduce human-like AI agents and evaluate the difficulty of levels through their play logs. These AI agents are designed to incorporate human nature, ensuring that difficulty evaluation aligns with human perspectives. Detailed

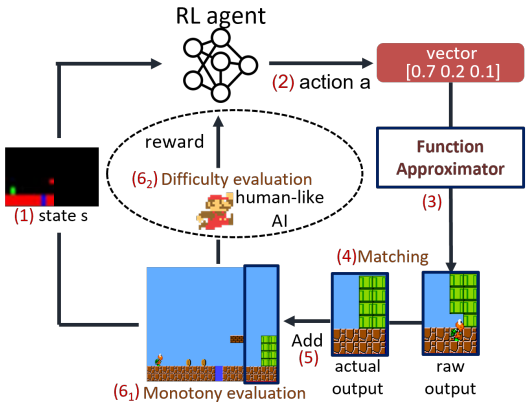



Fig. 3. The overall process of the level generation

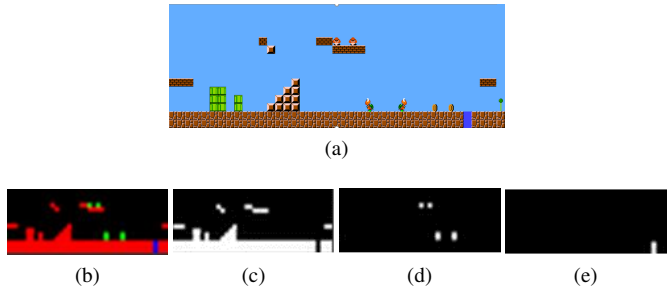



Fig. 4. An example of level representations. (a) is an example of a Super Mario level. (b) is an integrated image of (c), (d), and (e). (c) is a wall channel image, (d) is an enemy channel image, and (e) is a hole channel image.

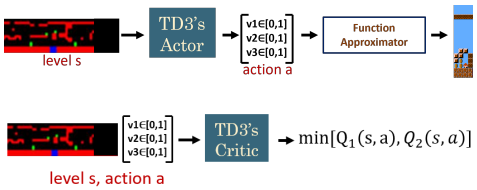



Fig. 5. TD3 for level generation in this study (top: actor, bottom: critic)

configuration of the human-like AI agents and the evaluation method are described in Sections IV-B3 and IV-D.

### A. Overall Level Generation Processes

The overall process of the level generation is shown in Fig. 3. (1) A level is transformed into a state (15×n matrix with three channels). (2) The RL agent generates an action represented as a vector instead of a pattern. (3) The vector is converted to the naturally connected pattern through the function approximator. (4) The mapped pattern is repaired through the Matching process. (5) The repaired pattern is sequentially added to the incomplete level. ($6_1$) Monotony is evaluated for intermediate actions. ($6_2$) Human-like AI agents evaluate the difficulty of the completed level.

### B. Markov Decision Process

This section explains the formulation of MDP for Super Mario level generation, encompassing state representation, action representation, state transition, and reward functions.

*1) State Representation (Incomplete Level):* Super Mario's level comprises intricate arrangements of walls, enemies, and holes. Various approaches can be employed to represent an incomplete level as a state for RL. Options include employing raw images of the level or utilizing a structured 15×n matrix consisting of 10 different tiles. We opt for a semantic reduction of this 10-tile matrix into three binary matrix channels representing walls, enemies, and holes. This method is particularly suitable for CNN to comprehend the state and generate actions effectively. The goal information is excluded since it is always on the right-most side.

To illustrate, an example Super Mario level depicted in Fig. 4(a) is transformed into the binary channel representation showcased in Fig. 4(b). In this representation, each color channel—red, green, and blue—corresponds to walls (c), enemies (d), and holes (e), respectively.

Both complete and incomplete levels are a series of patterns represented as a matrix of the same size. Incomplete patterns are depicted as vectors filled with -1, while those generated by the function approximator are represented as vectors filled with 0.

*2) Action Representation and State Transition:* The RL model generates an action as a vector composed of three real values within the $[0, 1]$ range. This vector is then transformed into a 15×4 pattern, consisting of 3 channels, as illustrated in Fig. 4 through the function approximator. For the function approximator, the CGAN model that considers natural connectivity between patterns [14] is used. CGAN takes the vector as input and outputs the corresponding pattern. The generated pattern is sequentially integrated into the level matrix. The generation process terminates when the level is filled by patterns.

Since the function approximation requires a vector of real values as input, we adopt the TD3 [63]. The actor component of TD3 takes the incomplete level matrix as input and produces a three-value vector within the $[0, 1]$ range. Then, the critic model evaluates the Q-value of the state-action pair, as shown in Fig. 5.

*3) Reward Function:* A level evaluation function is defined and used as the reward function of the MDP, encompassing three quality criteria: playability, difficulty, and monotony. High evaluation scores indicate high-quality levels.

First, playability is assessed using a strong A* AI agent implemented by Baumgarten [64]. A level is considered unplayable if this agent cannot complete it. Difficulty or monotony evaluation is omitted if the level is not playable.

Second, human-like AI agents, designed to mimic human-like mistakes, evaluate the level's difficulty. A new concept of *virtual damage* is introduced to evaluate the level's difficulty for human players. The total damage ($total_{dmg}$) is calculated using Eq. (1), which is a summation of the virtual damage inflicted by enemies ($enemy_{dmg}$) and holes ($hole_{dmg}$). It is assumed that jumping over a hole to avoid damage requires more movement than avoiding an enemy due to their size difference, especially when the hole is relatively large. Therefore, we assumed that avoiding damage inflicted by falling into a hole is more challenging for players than avoiding enemies. As

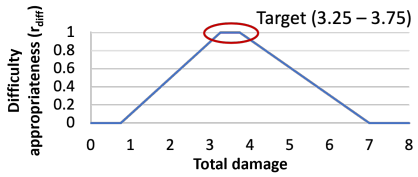



Fig. 6. Difficulty appropriateness ($r_{diff}$) based on the total damage ($total_{dmg}$)

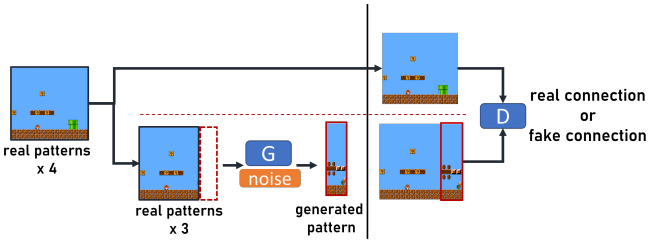



Fig. 7. A structure of CGAN for Super Mario pattern generation

a result, the $hole_{dmg}$ is assigned a slightly higher coefficient of 1.1. These coefficient settings were obtained through empirical trial and error tests on the game.

$$total_{dmg} = enemy_{dmg} \cdot 1 + hole_{dmg} \cdot 1.1 \quad (1)$$

The difficulty is assessed through multiple trials with human-like AI agents to capture various player behaviors. For each trial, the agent takes a different trajectory of actions. The details of human-like AI agents are explained in Section IV-D. The average value of the calculated $total_{dmg}$ represents the level's difficulty, with higher values indicating greater difficulty. The difficulty appropriateness ($r_{diff}$) is then computed using the function shown in Fig. 6, where the y-axis represents difficulty appropriateness ($r_{diff}$) associated with the total damage value. The rationale behind this function is to ensure that the generated levels are neither overly easy nor too difficult.

Lastly, monotony corresponding to the gameplay trajectory is evaluated. The decision of action sequences in a pattern is mainly affected by the layout of wall and hole tiles. Levels with the same adjacent patterns can lead to repetitive action decisions and monotonous gameplay. Monotony is evaluated by comparing the layout of wall and hole tiles in the current pattern ($p_n$) with those in the previous four patterns ($p_{n-1}$, $p_{n-2}$, $p_{n-3}$, $p_{n-4}$). A higher penalty is assigned to patterns that share wall/tile placements with their adjacent counterparts. For instance, if $p_n$ has the same wall/tile placements as both $p_{n-1}$ and $p_{n-2}$, the penalty assigned to $p_{n-1}$ is twice as much as that of $p_{n-2}$. This measure ensures that adjacent patterns result in different gameplay trajectories. Let $id(p, q)$ be a function that $id(p, q) = 1$ if and only if two patterns, p and q, have the same wall and hole tiles. Then, the monotony evaluation value ($r_{mono}$) is computed using Eq. (2).

$$r_{mono} = 1 - (8/15 \cdot id(p_n, p_{n-1}) + 4/15 \cdot id(p_n, p_{n-2}) + 2/15 \cdot id(p_n, p_{n-3}) + 1/15 \cdot id(p_n, p_{n-4})) \quad (2)$$

Monotony is assessed after generating a pattern, whereas difficulty evaluation occurs once the level is entirely generated. The weight assigned to the monotony evaluation value ($w_{mono}$) is specified as 0.01 due to its nature as an intermediate action.

### C. Function Approximator: From a Vector to a Pattern

CGAN is employed as the function approximator to map a vector to the corresponding pattern, considering the natural connection from previous patterns. Natural connectivity implies that subsequent patterns maintain similar features in appearance to the preceding ones. CGAN inputs a vector and the three previous patterns as a condition and outputs the subsequent naturally connected pattern as shown in Fig. 7. The vector is the action output of the RL agent, and the corresponding pattern is fixed through the matching and is added to the current incomplete level. Matching is an algorithm that selects the closest pattern by comparing the output noise pattern with all existing pattern data based on Euclidean distance. By using this function approximation, the generated levels ensure natural connectivity. The details of this function approximator and repairing its outputs through matching are readily available in our previous work [14], where a more comprehensive explanation is provided.

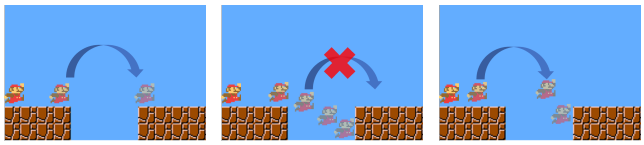
### D. Human-like AI Agents for Evaluation

We employ human-like AI agents to assess the difficulty of the generated levels, and the evaluation results serve as rewards for our RL model. We consider two aspects of human-like failures: input timing and input time span.

By nature, human players usually try to take specific actions to avoid dangers. However, input timing to execute actions in a given situation may be delayed or hastened due to failure caused by inaccuracies of human nature. Fig. 8 shows examples of failures caused by input timing of the jump. Mario needs to jump over the hole to avoid falling (a). Due to the delay (b) or hasten (c) in the jump input, Mario falls into the hole. These kinds of inaccuracies are prevalent in Super Mario. In addition, human players may sometimes press buttons longer than necessary.

From these human-like aspects, we can estimate the difficulty of the generated level as perceived by human players. Given that humans exhibit diverse gameplay behaviors, our human-like AI agents mimic these stochastic behaviors. In each episode, we utilized multiple AI agents to simulate the level, with each agent following a different action trajectory. This approach ensures that the AI agents can effectively represent variations in human gameplay styles. Here are the key details of our human-like AI agents:

- Our human-like AI is derived from the A* AI agent but with a focus on mimicking human behavior. In contrast to the original A* agent, which prioritizes efficiency and minimal jumping, our agents are encouraged to jump over obstacles like walls when possible. This is to mimic humans' tendency to jump over walls in Super Mario gameplay to obtain items placed on the walls or vertical



(a) (b) (c)

Fig. 8. Examples of ideal movements by humans (a) and failures due to the inaccuracies of human nature (b)-(c)

TABLE III
MEAN AND STANDARD DEVIATION OF THE NUMBER OF INDICATORS OF AI AGENTS' HUMAN-LIKE BEHAVIORS AT EACH LEVEL

| Level | 1-1 | 1-2 | 5-1 | 8-1 | 8-2 |
|---|---|---|---|---|---|
| Original A* jump (times) | 14.6 (±1.01) | 10.2 (±0.6) | 12 (±0) | 16.9 (±2.7) | 17 (±0) |
| Jump count (times) | 33 (±7.2) | 27.1 (±2.4) | 19 (±2.6) | 29.4 (±11.4) | 27.9 (±4.0) |
| $enemy_{dmg}$ (times) | 3.5 (±1.5) | 4.3 (±1.0) | 5.1 (±1.3) | 3.3 (±1.6) | 0.7 (±0.9) |
| $hole_{dmg}$ (times) | 1.5 (±1.0) | 1.6 (±0.5) | 2 (±1) | 4.5 (±2.5) | 7.4 (±2.0) |
| Long press (seconds) | 36.5 (±6.9) | 30.3 (±1.7) | 25.8 (±2.2) | 35.9 (±14) | 30.2 (±3.4) |

spaces. Humans also tend to constantly jump to avoid enemies or holes. To encourage jumping, we introduce additional rewards.

- To account for delayed or hastened input timing, the AI agent simulates virtual damage occurrences that could potentially be caused by input timing inaccuracies, referred to as virtual damage. This simulation occurs randomly (30%) and limitedly once per pattern. It is important to note that this simulation does not actually happen in actual gameplay. For instance, if an action is taken at frame $t$, we virtually perform the same action at frames $t+1$ and $t+2$ (delayed action) or frames $t-2$ and $t-1$ (hastened action). If the A* search indicates damage, it corresponds to scenarios depicted in Fig. 8(b) and Fig. 8(c), respectively. The locations where virtual damage occurs represent potential damage caused by input timing inaccuracies. The frequency of these occurrences is used in Eq. (1) to calculate $total_{dmg}$.
- Our human-like AI agents are also modified to occasionally execute the same actions for the next two frames instead of relying on the outcome of an A* search. Long presses also occur randomly (30%) and limitedly twice per pattern. The configuration of AI agents was selected based on empirical testing.

The data presented in Table III illustrates the average count and standard deviations of behaviors exhibited by the original A* agent and human-like AI agents across 10 trials, each conducted on 5 different reproduced Super Mario original levels. These results highlight the variability observed among trials, reflecting distinct player behaviors for each trial.

Moreover, Fig. 9 illustrates the reproduced Super Mario original level along with the locations of virtual damages observed in 50 AI agent trials. It is important to note that some virtual damages may overlap and are presented separately. These virtual damages consistently coincide with risky spots for human players within the levels. Consequently, virtual damages serve as reliable indicators of level difficulty.

## V. PCGRL CONSIDERING QUALITY AND DIVERSITY

Our PCGRL technique emphasizes enhancing the quality and diversity of the generated Super Mario levels to ensure an engaging gameplay experience. Below, we describe the methods used to achieve these objectives.

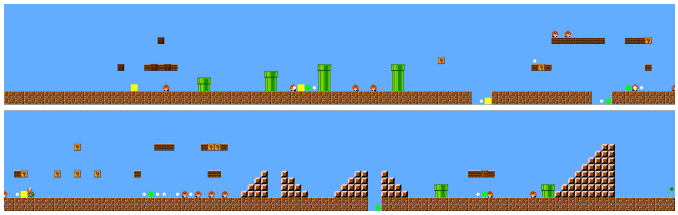


Fig. 9. The original level (1-1) with virtual damages indicated by symbols (yellow square: $dmg \geq 20$, green diamond: $10 \leq dmg < 20$, white circle: $dmg < 10$) from 50 AI agent trials

### A. Method for Quality Enhancement

We proposed an evaluation function that yields small evaluation values, typically ranging from 0 to 0.01, for intermediate actions. Further details on this function are provided in Section IV-B3. However, CAP arises from sparse reward signals from the small range of evaluation values. To effectively address this challenge, we have integrated virtual simulation (VS), a technique proven effective in handling CAP and enhancing content quality, as demonstrated in our previous work [15].

In this approach, RL virtually generates multiple completed levels for each intermediate action using the current policy, along with an exploration policy. The average evaluation of these virtually simulated levels is then used as the immediate reward for the intermediate actions. The experimental details and settings are given in Section VI-A2.

### B. Methods for Diversity Enhancement

Diverse content is a fundamental aspect of PCG, aiming to offer players fresh and engaging experiences while extending the longevity of games. Consequently, PCG methods should prioritize generating diverse and enjoyable levels rather than solely focusing on producing the single "best" one. One approach to enhancing diversity is incorporating it as a factor within the reward function.

However, our designed evaluation only focuses on quality criteria of individual levels, and simultaneous comparison of evaluation values of multiple levels can be complicated. Rather than altering the evaluation function, we propose two alternative methods to enhance diversity:

*1) Randomized Initialization:* We employ the RI technique introduced in our previous work. This approach introduces randomness into multiple starting patterns for each level rather than using an entirely empty level. Our previous findings demonstrated that this approach effectively balances the trade-off between quality and diversity.

Quality and diversity often exist in a trade-off relationship. For instance, if patterns are not randomly generated at the initial stage, RL tends to seek the "best" actions (patterns) and generates only the highest-quality levels, potentially compromising diversity. Conversely, if all patterns are randomly assigned, diversity may increase, but the overall quality of generated levels can decrease.

Our RI technique randomly initializes several patterns and lets the RL model complete the remainder of the level. The key parameter in this approach is the number of randomly initialized patterns, which directly impacts the balance between level quality and diversity. Unlike the change percentage technique employed in [52], both their method and ours may reduce the step size of the episode. Change percentage restricted changes from the initial levels. In contrast, our approach encourages more diversity by introducing randomness to the initial state, as the episode results depend on this initial condition. Comprehensive results illustrate this trade-off in Section VI-B.

*2) Diversity-Aware Greedy Policy:* Diversity obtained from RI is insufficient since it only depends on the initial states. We propose the DAGP as a subsequent method to further enhance diversity. This approach selects *not-bad-but-distant* actions based on Q-values, unlike the greedy policy that always opts for the best actions. By accepting slightly worse actions, DAGP introduces controlled stochastic noise into level generation, resulting in increased diversity. As a result, DAGP can generate relatively different levels with high quality even when starting with the same initial patterns.

The key to DAGP is the selection of *not-bad-but-distant* actions. Improper action selection can lead to either low evaluation scores or levels that are too similar. We present the noise introduction algorithm to select *not-bad-but-distant* actions as follows:

i) Determine the length of the level and train the RL model using Ornstein–Uhlenbeck (OU) noise [16] with $\epsilon$-greedy policy.

ii) Decide the location and number of noise patterns, which can be predetermined or randomly selected during generation. Then, specify the number of candidate patterns ($n$) and the KL-divergence ($kl_{pattern}$) to serve as the difference criteria between a candidate pattern and the best pattern.

iii) Patterns that are not noise are generated using the greedy policy. The action selection within noise pattern generation is depicted in Fig. 10. Start by generating $n$ random candidate actions. Exclude candidates within the gray area due to their similarity to the best action. The distance between two patterns is measured using KL-divergence. Candidates within the gray area (where $kl_{pattern}$ is less than the criteria value ($d$) indicate similarity to the best action. The size of the gray area ($d$) and the number of candidates ($n$) are adjustable hyperparameters. In this paper, experiments under several settings were tested in Section VI-B. Among the remaining candidates, select the one with the highest Q-value. The not-bad-but-distant candidate is the one that is not similar enough to the best action and is of sufficient quality (Q-value). If all candidates are rejected, the action is greedily selected.

## VI. Experiments and Discussions

The experiment was conducted using the following hardware components: GPU (GeForce GTX 1070), CPU (Intel Core i7-7700 3.60GHz), and RAM (16GB).


Fig. 10. Example of choosing an action that is *not-bad-but-distant* by DAGP

### A. High-quality Level Generation

We used TD3 to generate Super Mario levels with the PCGRL method and employed VS to address the CAP to enhance the level quality. Quality evaluation was conducted by comparing levels generated with TD3 alone and TD3 with VS (VS-TD3). RL rewards were considered both monotony and difficulty. The total episode reward ($r_{total}$) was calculated using Eq. (3).

$$r_{total} = r_{diff} + w_{mono} \sum r_{mono} \tag{3}$$

Below are the generation processes and results.

*1) Preparation of Patterns:*

- 10 original Super Mario levels were reproduced using Python Mario.
- Additional 12 custom levels were manually generated by humans.
- Each level was divided into multiple 15×4 patterns.
- A total of 6,232 patterns ($Pat_{legal}$) were extracted using a sliding window algorithm with a sliding size of 1.
- For a level L with length $n$, we obtained $n-3$ patterns $PL_1$ (x=1..4), $PL_2$ (x=2..5), .., $PL_{n-3}$.

*2) Experimental Setup:*

- The network settings of TD3 is noted in Table IV, where Conv $x$ means a convolutional layers with $x$ filters of size $3\times3$,Pool means a max pooling with pool size $2\times2$, and FC $x$ means a fully-connected layers with $x$ nodes. The layer composition was set up according to the general manner [65] of the CNN.
- Level length was set to 41. Each level was represented by a 15×40 matrix, requiring generated 10 patterns with the last tile reserved for the goal.
- Three imaginary previous patterns ($ip_1$, $ip_2$, $ip_3$) in Fig. 11 (1) were given to CGAN to generate the first pattern ($p_1$) in Fig. 11 (2). Imaginary patterns are selected by a series of three consecutive patterns from $Pat_{legal}$. Then, next pattern ($p_2$) is generated using $ip_2$, $ip_3$, and $p_1$. These imaginary patterns are only used for generating the first three patterns.
- The first two patterns ($p_1$, $p_2$) from Fig. 11 (2) were generated by inputting random noises to CGAN and applying matching and used as the initial states.
- With two patterns randomly initialized, the RL actor generates the remaining 8 patterns (see Fig. 11 (3)).

TABLE IV
TD3 SETUP

| Parameter | Value |
|---|---|
| Layers | Conv 32→32→Pool→<br>64→64→Pool→<br>FC 256→128→128 |
| Memory size | 40000 |
| Learning rate | Actor:$5 \times 10^{-6}$<br>Critic: $5 \times 10^{-5}$ |
| Target network | soft update ($5 \times 10^{-5}$) |
| Batch size | 32 |
| Discount factor | 0.99 |
| Exploration | OU noise with $\epsilon$-greedy<br>($\epsilon$ 0.5 $\rightarrow$ 0.1 at 30% of episodes) |

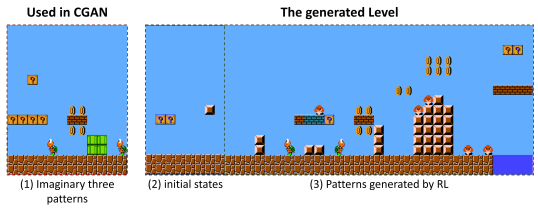
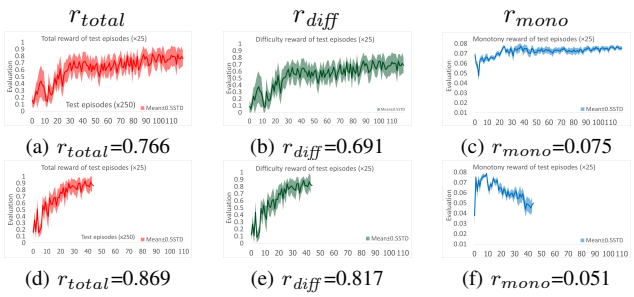



Fig. 11. Imaginary patterns and generated patterns

- VS was applied at the beginning of the training, with the number of episodes set to 5. This means that 5 stages are generated for each intermediate action. Since there were 7 intermediate actions per episode, excluding the last action, training took approximately 36 times longer per episode. Each evaluation involved 10 trials of AI agents. A total of 360 trials of simulation from AI agents were conducted.

*3) Training Result:* TD3 and VS-TD3 were trained for 160 hours and 168 hours, respectively. Within this timeframe, TD3 was trained for 30,000 episodes, while VS-TD3 was trained for 1,100 episodes. The learning curves are shown in Fig. 12. Fig. 12 (a)-(c) illustrate the average evaluations during the test phase for TD3 while Fig. 12 (d)-(f) show the same for VS-TD3. In each set of three graphs, the red, green, and blue curves correspond to $r_{diff}$, $r_{mono}$, and $r_{total}$, respectively.

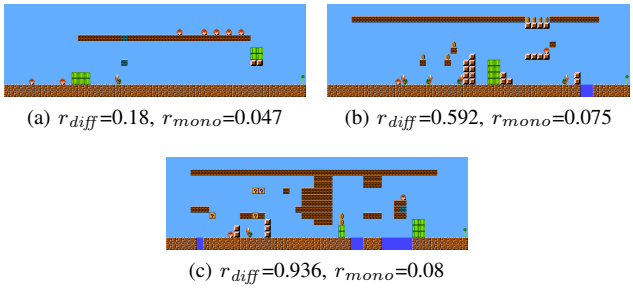
In Fig. 12 (a)-(c), we present the evaluation values and standard deviations of 25 levels during the test phase of TD3. The increasing evaluation values demonstrate the RL model's ability to learn level generation. Fig. 12 (d)-(f) shows the evaluation values and standard deviations of 5 levels during the test phase of VS-TD3. VS-TD3 was trained for fewer episodes due to the longer time per episode. Despite this, VS-TD3 achieved better rewards than TD3 in the early stages of training. During testing, 2 out of 50 levels generated by TD3 and 2 out of 50 levels generated by VS-TD3 were unplayable, indicating a need to recreate those levels for practical use. Due to the lengthy duration required for each episode in VS-TD3 training, relatively few episodes were conducted for fair comparison. Nevertheless, VS-TD3 obtained a better reward than TD3 in the very early training. In many training trials, the difficulty evaluation $r_{diff}$ tends to be trained first, with the monotony evaluation $r_{mono}$ trained subsequently. Though $r_{mono}$ is decreasing in the early training phase for VS-TD3, it is expected to have a higher $r_{mono}$ value if more time is given.




(a) $r_{total}$=0.766 (b) $r_{diff}$=0.691 (c) $r_{mono}$=0.075

(d) $r_{total}$=0.869 (e) $r_{diff}$=0.817 (f) $r_{mono}$=0.051

Fig. 12. Average level evaluations: (top) TD3's performance in a 25-trial test phase for the 10-pattern level. (bottom) VS-TD3's performance in a 5-trial test phase for the 10-pattern level. Each value in all curves indicates the average reward of the last 20% episodes.



(a) $r_{diff}$=0.18, $r_{mono}$=0.047 (b) $r_{diff}$=0.592, $r_{mono}$=0.075

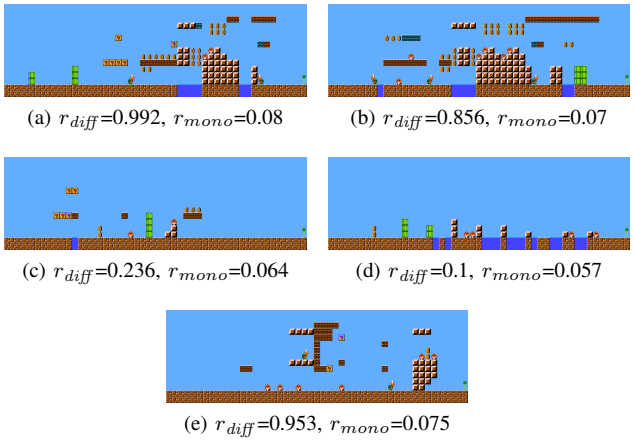
(c) $r_{diff}$=0.936, $r_{mono}$=0.08

Fig. 13. Examples of levels generated during (a) early (b) mid (c) end of training

Fig. 13 shows the levels generated during the training of TD3, sampled from Fig. 12 (a)-(c). Fig. 13 (a) represents a level generated in the early training phase, exhibiting low difficulty (0.18) and monotony evaluation value (0.047). As training progressed, the evaluation of monotony was learned first. The level in Fig. 13 (b) becomes relatively difficult (0.592) with a high monotony evaluation value (0.075). Then, TD3 generated a level in Fig. 13 (c) that is appropriately difficult (0.936) and non-monotonous (0.08).

Furthermore, we compared the time required to achieve high-quality levels between random generation, TD3, and VS-TD3. For the random generation, it selects random patterns from $pat_{legal}$. Random generation involves the generate-and-test process, which takes 20 seconds and requires approximately 40 trials to obtain a level with a high evaluation value. Meanwhile, both TD3 and VS-TD3 could generate a level with an expected average evaluation value of 0.8 within 1-2 seconds, including playability assessments. This indicates that random generation is not insufficient for producing a high-quality level on demand. Our approach, on the other hand, successfully generates a level with quality on demand.

*4) Generated Levels:* Fig. 14 illustrated examples of levels generated using three methods. Levels (a)-(b) are generated by TD3, levels (c)-(d) by inputting random vector to the function approximation, and level (e) by random generation. Levels (a)-(b) are non-monotonous with distinct patterns, with proper layout and number of holes and enemy tiles to ensure adequate



(a) $r_{diff}$=0.992, $r_{mono}$=0.08 (b) $r_{diff}$=0.856, $r_{mono}$=0.07

(c) $r_{diff}$=0.236, $r_{mono}$=0.064 (d) $r_{diff}$=0.1, $r_{mono}$=0.057

(e) $r_{diff}$=0.953, $r_{mono}$=0.075

Fig. 14. Examples of levels generated by TD3 (a)-(b), the process of inputting random vectors into the function approximator (c)-(d), and the selection of random patterns (e)

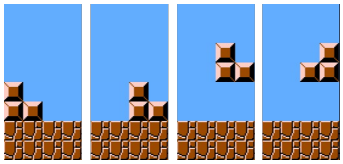


Fig. 15. Four patterns with three blocks

TABLE V
COMPARISON OF QUALITY AND DIVERSITY INDICATORS FROM 500 LEVELS GENERATED WHEN THE FIRST $c$ PATTERNS WERE RANDOMLY INITIALIZED.

| $c$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $KL_{level}$ | 1.569 (±0.812) | 1.817 (±0.734) | 1.939 (±0.686) | 1.911 (±0.644) |
| Different tiles | 89 (±48) | 100 (±42) | 107 (±42) | 105 (±39) |
| Reward | 0.728 (±0.365) | 0.687 (±0.392) | 0.685 (±0.377) | 0.670 (±0.394) |
| $c$ | 5 | 6 | 7 | 8 |
| $KL_{level}$ | 1.979 (±0.644) | 2.025 (±0.626) | 2.000 (±0.626) | 1.939 (±0.570) |
| Different tiles | 110 (±41) | 111 (±39) | 110 (±40) | 107 (±37) |
| Reward | 0.680 (±0.377) | 0.589 (±0.384) | 0.547 (±0.346) | 0.493 (±0.343) |

difficulty. In contrast, level (c) has too few holes and enemies, and level (d) has too many holes. Thus, their evaluation values are low. Although level (e) exhibits a high evaluation value of 0.953, each pattern is independent and shares no apparent connection. This is because random generation does not have the function approximation that ensures natural connections among patterns.

The experimental results showcase the capability of our PCGRL method to consistently produce Super Mario levels that receive high evaluations in terms of non-monotony and proper difficulty as perceived by human players. In addition, the generated levels also exhibit natural connections of patterns.

### B. Diverse Level Generation

Following the successful generation of high-quality Super Mario levels, our next step was to obtain diverse levels while maintaining quality. To achieve this, we employed RI and DAGP.

*1) Diversity Assessment:* The diversity of levels is assessed using two indicators as follows:

- The number of different tiles between the two levels: This is a straightforward way to evaluate the differences between levels.
- The KL-divergence between two levels ($KL_{level}$): Each wall and enemy tile are converted to the value of 2 and 3, respectively. Then, KL-divergence of (40×15)-dimensional vectors of two patterns are calculated. The value of $KL_{level}$ between two levels indicates a comparison of the distribution of tile arrangement. Low $KL_{level}$ value means both levels have similar wall and enemy arrangements, even if the tiles are different.

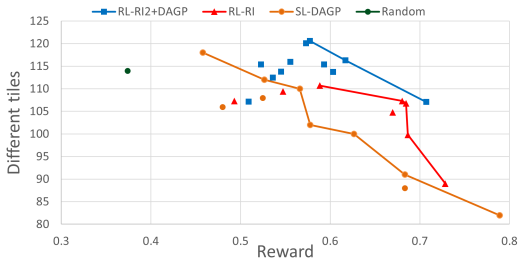
Fig. 15 shows four patterns with three blocks. Compared to the first pattern, other patterns have the same number of different tiles, which is 6. However, players may feel that the respective second and fourth patterns are the most and least similar to the first pattern. The $KL_{level}$ value can help to distinguish these similarities.

*2) Result:* Generation of diverse levels generally requires a well-trained model. Thus, we employed VS-TD3, the model used to generate high-quality levels. First, RI was used to generate 500 levels, each integer $c \in [1, 8]$, where the first $c$ patterns were randomly generated. Table V shows the average $KL_{level}$, numbers of different tiles and rewards for each $c$. Note that the levels contained 10 patterns. As expected, the result demonstrated the increasing trend of diversity while the average rewards decreased when more patterns were randomly initialized.

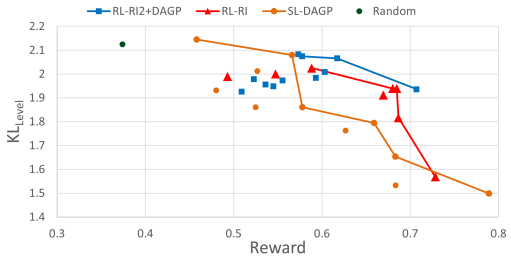
Second, a combination of DAGP and RI was used. RI was used to generate the first two patterns, and DAGP was applied to generate $4^{th}$, $5^{th}$, and $6^{th}$ patterns. In DAGP, $n$ candidates were generated and candidates with at least $d$ away from the actor's action were gathered. KL-divergence between patterns ($KL_{pattern}$) is used as a distance threshold $d$. Then, DAGP selected the one with the highest Q-value. The influence of DAGP is investigated by generating levels from the same initial levels used in RI. Several configurations of $d$ and $n$ were tested, and 500 levels were generated from each setting. The DAGP settings of $d$=0.05, 0.1, 0.2, and $n$=5, 10, 20, 40 are used.

We compared the quality and diversity of level generation among different approaches: applying only RI (RL-RI), RI with DAGP (RL-RI2+DAGP), Supervised learning with DAGP (SL-DAGP), and random generation. For SL-DAGP, we prepared two network models similar to the actor-critic model, having the same structures as TD3, except they have a batch normalization layer [66] to prevent overfitting. Random generation is employed to prepare training data for SL. Two types of data are collected: 1) levels above 0.7 evaluation values for training the action. 2) levels covering the full evaluation range for training the critics. The critic's training data should contain levels with all ranges of evaluation to better predict the evaluations of state-action pairs. It took 27 and 5 hours to obtain 10,000 levels, respectively.

Fig. 16 displays the Pareto frontier of the average rewards (x-axis) versus different tiles and $KL_{level}$. Our findings indicate that while random generation can produce diverse levels,




(a) The number of different tiles




(b) $KL_{level}$

Fig. 16. Quality-diversity comparison between applying only RI, RI with DAGP, and random generation

(a) $r_{diff}$=0.916, $r_{mono}$=0.075

(b) $r_{diff}$=0.824, $r_{mono}$=0.075

(c) $r_{diff}$=0.840, $r_{mono}$=0.073

Fig. 17. Levels generated by the actor policy (a) and DAGP (b, c) with the same first two patterns (bottom)

it is inefficient in generating high-quality content. RL-RI and SL-DAGP, on the other hand, demonstrate the capability to generate quality levels, although diversity is compromised. However, the combination of RL-RI2+DAGP shows its superiority in the generation of high-quality and diverse levels.

*3) Generated Levels:* Fig. 17 shows examples of levels generated by VS-TD3 (a) and DAGP (b)-(c) with the same first two patterns. Compared to level (a), which represents the level with the highest evaluation score, levels (b)-(c) still have relatively high evaluation scores with good diversity indicators of $KL_{level}$ being 1.748 and 1.586, and the number of different tiles being 126 and 103, respectively. These results demonstrate that by applying DAGP, the RL model can maintain high-quality level generation while introducing greater diversity, as evidenced by the relatively high evaluation scores and diversity indicators in levels (b) and (c).

### C. Human Subject Evaluations

We further conducted a questionnaire on human subjects to know whether our criteria are well reflected or not based on the Likert scales. Participants evaluated the levels based on our defined criteria, including natural connection, difficulty, monotony, and diversity. Responses obtained from 33 participants are shown in Fig. 18. From Fig. 18 (a), 5 levels from random generation (R) and 5 levels from inputting random vectors to the function approximator (V) were subjects of evaluation. Levels R1-R5 did not consider the natural connectivity, whereas V1-V5 levels considered the natural connection from previous patterns. The result shows that levels generated by the function approximator seem more naturally connected in human perceptions.

From Fig. 18 (b)-(c), 24 levels were gathered from random generation, inputting random vectors to the function approximator, DAGP, to evaluate our difficulty (b) and monotony (c) criteria. Note that a 7-point scale is employed in assessing difficulty, allowing for a clear distinction between the perception of whether the levels are challenging, properly difficult, or easy. Because our difficulty criterion value ($r_{diff}$) are low for levels that are too easy or too difficult, we employed a *difficulty value* as shown in Eq. (4) to distinguish between easy and difficult levels. Pearson correlation coefficients are 0.805 and 0.693 for difficulty and monotony, respectively, indicating our evaluations are highly correlated to human perceptions of the levels.

$$Difficulty\ value = \begin{cases} \mathrm{r}_{diff}, & \text{if } total_{dmg} \leq 4 \\ \text{2-r}_{diff}, & \text{otherwise} \end{cases} \quad (4)$$

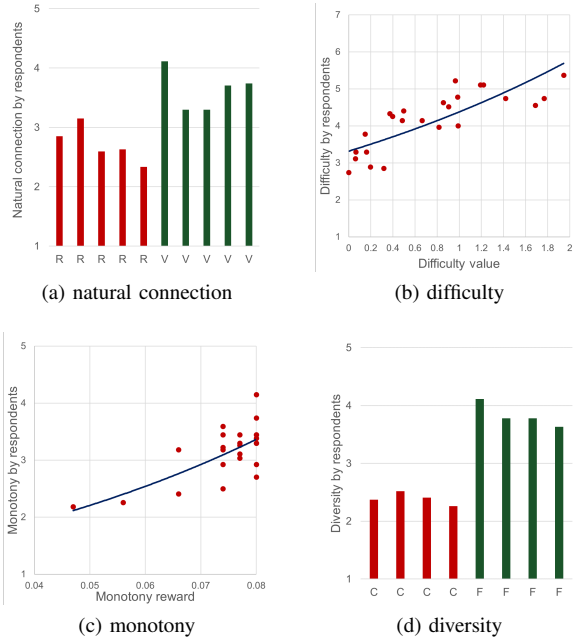
Lastly, from Fig. 18 (d), 8 sets of evaluation subjects, each consisting of 3 levels, were assessed to verify whether $KL_{level}$ is a suitable diversity indicator. Levels in four sets (C) exhibit low $KL_{level}$ values among them, indicating similar tile arrangements, while levels in the other four sets (F) have high $KL_{level}$ values, suggesting diverse tile arrangements among levels. Participants' evaluations indicate that level sets with high $KL_{level}$ values (F) are perceived as more diverse, whereas sets with low $KL_{level}$ values (C) are less diverse, demonstrating a positive correlation between $KL_{level}$ and human perception of diversity.

## VII. Conclusion

In this paper, we introduce a novel PCGRL technique technique for generating diverse and high-quality Super Mario levels. Our approach formulates the level generation problem into an MDP. One primary key to this work is the development of an evaluation function that encompasses critical quality criteria, including playability, difficulty, and monotony, to ensure game longevity and engagement.

We addressed challenges posed by the high dimensionality of RL states and actions by employing function approximation, which not only mitigated dimensionality issues but also facilitated natural connectivity among level patterns. Additionally, we tackled the challenge of assessing difficulty from a human perspective by introducing human-like AI agents that emulate human inaccuracies. To enhance the quality and diversity, we introduced three novel methods: VS, RI, and DAGP.

Our experimental results demonstrate the efficacy of our PCGRL approach in producing promising Super Mario levels with high quality and diversity. These findings lay a foundation for advancing game development and procedural content




(a) natural connection (b) difficulty

(c) monotony (d) diversity

Fig. 18. Results from human subject evaluations based on levels generated by the proposed method, verifying natural connection, difficulty, monotony, and diversity

generation research, contributing innovative solutions to existing challenges in the field. We believe that the concept of formulating PCG into MDP and applying RL is a generalizable concept applicable to various games. As long as the PCG problem can be formulated as an MDP, RL algorithms can then be applied to learn good policy (i.e., what actions to take for given states) based on a specified reward function. It is also important to note that different MDP formulations and designing a reward function may be required to address challenges emerging from the nature of other games. After learning, the policy serves as the content generator at relatively low generation costs. Furthermore, human subject experiments were conducted to assess how actual players perceive our criteria, including natural connection, difficulty, monotony, and diversity.

While our results confirmed the efficacy of our proposed method, we acknowledge its limitations due to the stochastic behavior of human-like AI agents. This can result in variations in evaluation values, leading to a lack of high positive correlations between Q-values and evaluation scores, potentially affecting the performance of DAGP. To overcome this limitation, future research endeavors could focus on stabilizing evaluation values by increasing the number of trials or exploring alternative strategies. Furthermore, our work focused on evaluating difficulty and monotony as quality criteria. Other essential aspects, such as the placement of coins and mushrooms, can also impact the entertainment value of Super Mario levels. These potential enhancements will not only refine the PCGRL technique but also enrich the gaming experience for players and contribute to the ongoing evolution of PCG methodologies.

## ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Number JP23K11381.

## REFERENCES

[1] E. Hauck and C. Aranha, "Automatic generation of super mario levels via graph grammars," in *2020 IEEE Conference on Games (CoG)*, 2020, pp. 297–304.

[2] A. Gellel and P. Sweetser, "A hybrid approach to procedural generation of roguelike video game levels," in *Proceedings of the 15th International Conference on the Foundations of Digital Games*, ser. FDG '20, 2020.

[3] J. Togelius, M. Preuss, and G. N. Yannakakis, "Towards multiobjective procedural map generation," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, ser. PCGames '10, 2010.

[4] G. Pickett, F. Khosmood, and A. Fowler, "Automated generation of conversational non player characters," in *INT/SBG@AIIDE*, 2015.

[5] D. Gravina and D. Loiacono, "Procedural weapons generation for unreal tournament iii," in *2015 IEEE Games Entertainment Media Conference (GEM)*, 2015, pp. 1–8.

[6] 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.

[7] Y. Tsujino and R. Yamanishi, "Dance Dance Gradation: A Generation of Fine-Tuned Dance Charts," in *Entertainment Computing – ICEC 2018*, Cham, 2018, pp. 175–187.

[8] 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.

[9] T. Morita and H. Hosobe, "Video game agents with human-like behavior using the deep q-network and biological constraints." in *ICAART (3)*, 2023, pp. 525–531.

[10] N. Fujii, Y. Sato, H. Wakama, K. Kazai, and H. Katayose, "Evaluating human-like behaviors of video-game agents autonomously acquired with biological constraints," in *Advances in Computer Entertainment*. Cham: Springer International Publishing, 2013, pp. 61–76.

[11] A. Summerville *et al.*, "Procedural content generation via machine learning (pcgml)," *IEEE Transactions on Games*, vol. 10, no. 3, pp. 257–270, 2018.

[12] 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.

[13] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural content generation*. Springer, 2016.

[14] S. Nam, C.-H. Hsueh, and K. Ikeda, "Procedural content generation of super mario levels considering natural connection," in *2023 20th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 2023, pp. 291–296.

[15] S. Nam, C.-H. Hsueh, and K. Ikeda, "Generation of game stages with quality and diversity by reinforcement learning in turn-based RPG," *IEEE Transactions on Games*, 2021.

[16] 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*.

[17] A. Tavakoli, F. Pardo, and P. Kormushev, "Action branching architectures for deep reinforcement learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018.

[18] T. Shu, J. Liu, and G. N. Yannakakis, "Experience-driven pcg via reinforcement learning: A super mario bros study," in *2021 IEEE Conference on Games (CoG)*, 2021, pp. 1–9.

[19] T. Shu, Z. Wang, J. Liu, and X. Yao, "A novel cnet-assisted evolutionary level repairer and its applications to super mario bros," in *2020 IEEE Congress on Evolutionary Computation (CEC)*, 2020, pp. 1–10.

[20] 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.

[21] 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*, 2012, p. 1–8.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] N. Shaker, M. Nicolau, G. N. Yannakakis, J. Togelius, and M. O'Neill, "Evolving levels for super mario bros using grammatical evolution," in *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, 2012, pp. 304–311.

[27] Z. Wang, J. Liu, and G. N. Yannakakis, "The fun facets of mario: Multifaceted experience-driven pcg via reinforcement learning," in *Proceedings of the 17th International Conference on the Foundations of Digital Games*, ser. FDG '22. New York, NY, USA: Association for Computing Machinery, 2022.

[28] P. Shi and K. Chen, "Learning constructive primitives for real-time dynamic difficulty adjustment in super mario bros," *IEEE Transactions on Games (TOG)*, vol. 10, no. 2, pp. 155–169, 2018.

[29] J. Flimmel, J. Gemrot, and V. Černý, "Coevolution of ai and level generators for super mario game," in *2021 IEEE Congress on Evolutionary Computation (CEC)*, 2021, pp. 2093–2100.

[30] J. Zhang, T. Gao, and Q. Mi, "Side-scrolling platform game levels reachability repair method and its applications to super mario bros," in *2022 RIVF International Conference on Computing and Communication Technologies (RIVF)*, 2022, pp. 232–237.

[31] M. Beukman, S. James, and C. W. Cleghorn, "Towards objective metrics for procedurally generated video game levels," *CoRR*, vol. abs/2201.10334, 2022.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

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

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] A. Sarkar, Z. Yang, and S. Cooper, "Controllable level blending between games using variational autoencoders," in *Proceedings of the EXAG Workshop at AIIDE*, 2019.

[44] 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.

[45] 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.

[46] F. Abraham and M. Stephenson, "Utilizing generative adversarial networks for stable structure generation in angry birds," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 19, no. 1, pp. 2–12, Oct. 2023.

[47] M. Siper, A. Khalifa, and J. Togelius, "Path of destruction: Learning an iterative level generator using a small dataset," in *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2022, pp. 337–343.

[48] M. Siper, S. Earle, Z. Jiang, A. Khalifa, and J. Togelius, "Controllable path of destruction," *arXiv preprint arXiv:2305.18553*, 2023.

[49] F. Schubert, M. Awiszus, and B. Rosenhahn, "Toad-gan: A flexible framework for few-shot level generation in token-based games," *IEEE Transactions on Games*, vol. 14, no. 2, pp. 284–293, 2022.

[50] P. Bontrager and J. Togelius, "Learning to generate levels from nothing," in *2021 IEEE Conference on Games (CoG)*, 2021, pp. 1–8.

[51] 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.

[52] 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.

[53] S. Earle, M. Edwards, A. Khalifa, P. Bontrager, and J. Togelius, "Learning controllable content generators," in *2021 IEEE Conference on Games (CoG)*, 2021, pp. 1–9.

[54] Z. Jiang, S. Earle, M. Green, and J. Togelius, "Learning controllable 3d level generators," in *Proceedings of the 17th International Conference on the Foundations of Digital Games (FDG)*, 2022.

[55] O. Delarosa, H. Dong, M. Ruan, A. Khalifa, and J. Togelius, "Mixed-initiative level design with RL brush," *CoRR*, vol. abs/2008.02778, 2020.

[56] Y. Zakaria, M. Fayek, and M. Hadhoud, "Start small: Training controllable game level generators without training data by learning at multiple sizes," *Alexandria Engineering Journal*, vol. 72, p. 479–494, Jun. 2023.

[57] O. Missura and T. Gärtner, "Player modeling for intelligent difficulty adjustment," in *Discovery Science*, 2009, pp. 197–211.

[58] M. Csikszentmihalyi and M. Csikszentmihalyi, "Toward a psychology of optimal experience," *Flow and the foundations of positive psychology: The collected works of Mihaly Csikszentmihalyi*, pp. 209–226, 2014.

[59] C. S. Ikehara, M. E. Crosby, and P. A. Silva, "Combining augmented cognition and gamification," in *Foundations of Augmented Cognition: 7th International Conference, AC 2013, Held as Part of HCI International 2013, Las Vegas, NV, USA, July 21-26, 2013. Proceedings 7*. Springer, 2013, pp. 676–684.

[60] R. Koster, *Theory of fun for game design*. O'Reilly Media, Inc., 2013.

[61] mx0c, *super-mario-python*, https://github.com/mx0c/super-mario-python, 2021.

[62] S. Karakovskiy and J. Togelius, "The mario ai benchmark and competitions," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 55–67, 2012.

[63] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 1587–1596.

[64] J. Togelius, S. Karakovskiy, and R. Baumgarten, "The 2009 mario ai competition," 08 2010, pp. 1–8.

[65] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into Imaging*, vol. 9, pp. 611 – 629, 2018.

[66] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.