Online Game Level Generation from Music

Ziqi Wang*, Jialin Liu†*
* Research Institute of Trustworthy Autonomous System
Southern University of Science and Technology (SUSTech)
†Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation
Department of Computer Science and Engineering
Southern University of Science and Technology (SUSTech)
Shenzhen, China

Abstract—Game consists of multiple types of content, while the harmony of different content types play an essential role in game design. However, most works on procedural content generation consider only one type of content at a time. In this paper, we propose and formulate online level generation from music, in a way of matching a level feature to a music feature in real-time, while adapting to players’ play speed. A generic framework named online player-adaptive procedural content generation via reinforcement learning, OPARL for short, is built upon the experience-driven reinforcement learning and controllable reinforcement learning, to enable online level generation from music. Furthermore, a novel control policy based on local search and k-nearest neighbours is proposed and integrated into OPARL to control the level generator considering the play data collected online. Results of simulation-based experiments show that our implementation of OPARL is competent to generate playable levels with difficulty degree matched to the “energy” dynamic of music for different artificial players in an online fashion.

Index Terms—Procedural content generation, online level generation, player-adaptive, EDPCG, EDRL

I. INTRODUCTION

Human perception is multi-modal. Digital games, as an emerging creation field, lies in the intersection of multiple types of content that meet different aspects of human perception, has the ability of expressing stories, emotions or aesthetics, and satisfying human’s natural entertainment demand [1], [2]. A successful game should guarantee the harmony of different types of content. Procedural content generation (PCG) [3]–[6], aiming at the automated or mixed-initiative creation of game contents, such as levels, maps, musics, rules and narratives, has shown its potential to reduce game development costs, augment the creativity of individual human creators, and provide personalised game contents [2], [6]. There have been extensive researches in generating a single type of game content [2], [6], while only a few works considered generating one type of content driven by another [7]–[9]. Procedurally generating a complete game of several content types is an emerging topic [6], [10]. For instance, AudioInSpace [10] generated game rules, visuals and audio in a mix-initiative way. The aforementioned works [1], [7]–[10] mainly focused on puzzle games, shooting games and rhythm games.

In this work, we focus on platformer games, in which levels and music together affect the player experience. We investigate into level generation from music, ideally to be achieved in an
online fashion for some games, assuming that the play experience should be consistent with the current music. For example, if the music in a platformer game is intense or nervous, the appeared level segments would be more difficult to match the atmosphere created by music. Some commercial games, such as Dance Dance Revolution (Konami, 1998), Guitar Hero (RedOctane, 2005) and Muse Dash (PeroPeroGames, 2018), directly force the gameplay to be consistent with music by designing specific game rules. However, for most games of other genres, it’s hard to generate levels from music because the rhythm of playing a level depends on the player’s play speed and thus not fixed, which will result in the failure of matching a level to music in an offline or player-unaware way. Fig. 1 illustrates why the play speed affects the online generation of level from music. The level segments on top and bottom are played by different artificial players at the same time window but have different length as the players played the levels in different speed. Moreover, the generation system should determine the difficulty degree of the next level segment in real-time to make sure that the segment will be played at the best-matched time slice. Therefore, online and player-adaptive level generation from music is desired.

In this paper, a framework for online level generation from music named online player-adaptive procedural content generation via reinforcement learning, OPARL for short, is proposed. OPARL follows a controller-generator architecture. The generator takes the current level segment and a music feature as input and outputs a new level segment with feature matched to the input at each iteration. When OPARL generates a level segment from music, the generator is controlled by a novel policy named local search with k-nearest neighbours based estimation (LS-KNN) which takes historical play data into account to produce the targeted feature value that minimises an inner error respect to an ideal feature sequence derived from the given music’s temporal feature sequence (detailed in section III). The controllable generator is extended from the experience-driven procedural content generation via reinforcement learning (EDRL) [11] framework by leveraging controllable reinforcement learning [12], [13]. The proposed OPARL framework is implemented and verified on the benchmark game for level generation, Super Mario Bros. (SMB). Experimental results show that the resulted system can generate playable SMB levels that are consistent with the given music\(^1\).

Our framework requires no expert knowledge except for the content representation, a few training examples for training a generative adversarial network (GAN) [14], [15] and the CNet-assisted repairer [16] for determining and repairing broken pipes. Although our proposed approach is verified in generating SMB levels from music, applying it to other platformer games, such as Megaman (CAPCOM, 1987), Electronic Super Joy (Michael Todd, 2014) and Celeste (Matt Makes Games Inc., 2018), is straightforward.

The remainder of this paper is organised as follows. Section II presents some related work. Section III formulates the problem of online level generation from music and addresses its challenges. Then, our OPARL framework and its technical details are presented in Section IV. In Section V, the effectiveness and robustness of OPARL to different players and music are verified through its implementation for SMB and simulation-based experiments. Section VI concludes and discusses some future directions.

II. BACKGROUND

This section discusses related work that involves multiple types of content and online player-adaptive approach.

A. PCG that Involves Multiple Content Types

Liapis et al. proposed the concept of orchestrating game generation, which aims at generating different types of game content jointly and harmoniously [1]. One representative orchestrating game generation system, AudioInSpace [10], generates game rules, visuals and audio in a mix-initiative way. Plans and Morelli proposed an experience-driven generator to generate music that reacts to the “excitement” of the game using search-based algorithms [7]. Naushad and Muhammad introduced a conditional music generation framework to enable adaptive music generation for games [8]. Engels et al. developed an hierarchical Markov model-based music generation system to produce music pieces in real-time [9].

The aforementioned works mainly focus on generating cosmetic content from functional content [5], while there are also research works that explored the reversed way. For instance, some works concentrated on learning to generate rhythms game charts from music via supervised learning [17], [18]. Jordan et al. introduced a mobile game named BeatTheBeat which applied self-organising maps method to create game levels that match some music features [19]. In the work of [20], a mixed-initiative PCG system is presented by Karavolos et al. to generate game levels from mission or space provided by human designers. Atmaja et al. proposed a top-down PCG framework to “translate” platformer games from storyline [21]. To our best knowledge, no work has ever generated platformer game levels from music in real-time.

B. Online Player-Adaptive Level Generation

A popular research topic related to online player-adaptive level generation is dynamic difficulty adjustment (DDA), aiming at adjusting levels’ difficulty degree considering the abilities or skills of players for desired player experience or aesthetic goal [22]. Shi and Chen proposed a DDA policy based on Thompson sampling, and embedded it into an online level generation framework [23]. Stammer et al. applied a conditional player experience model considering different play styles to generate player-adaptive Spelunky levels with DDA [24].

Some works built on experience-driven procedural content generation [25]. For instance, Shaker et al. applied player modelling and grammar evolution to generate online levels that optimise player experience [26]. Blom et al. generated online

\(^1\)Code, experimental data and demo of this paper are available on GitHub: https://github.com/PneuC/OPARL.
personalised SMB levels with facial expression recognition [27]. Shu et al. introduced the EDRL framework to enable real-time level generation with experience-driven reward functions as content quality measurements [11].

III. MUSIC-DRIVEN ONLINE LEVEL FEATURE CONTROL

To generate level segments that are consistent to a given piece of music in real-time, we consider ensuring the consistency or harmony by matching a feature of level segments to a feature of the given piece of music, referred to as music-driven online level feature control in this paper. The problem of finding online the optimal value of level feature that matches the given music is formulated in Section III-A. Then, the challenges of achieving online music-driven level feature control are discussed in Section III-B.

A. Problem Formulation

Given a piece of music, an ideal feature sequence of a level can be derived in some fine-grained time unit, denoted as \( \mathcal{F}^* = (f^*_1, \ldots, f^*_i, \ldots) \). Let \( f_i \) and \( \delta_i \) denote the target feature produced by the controller at the \( i \)th iteration and the duration of playing the \( i \)th segment, \( S_i \), respectively. The objective is to minimise the error defined as follows:

\[
\varepsilon_{\text{inner}} = \frac{1}{T(f_{\text{max}} - f_{\text{min}})} \sum_{i=1}^{N} \left( \sum_{t=b_i}^{b_{i+1}} |f^*_t - f_t| \right),
\]

where \( N \) is the number of segments and \( b_i = \sum_{k=0}^{i-1} \delta_k \) indicates the starting time of playing the level segment \( S_i \) with \( \delta_0 = 0 \). \( f_{\text{max}} \) and \( f_{\text{min}} \) are the upper bound and lower bound of the level feature, respectively. \( T = \sum_{i=1}^{N} \delta_i \) is the total time spent to play through the whole level. The decision space of online music-driven level feature control is \([f_{\text{min}}, f_{\text{max}}]\).

This error \( \varepsilon_{\text{inner}} \) is called an inner error because Eq. (1) evaluates the distance between the targeted feature values and the ideal feature sequence, and cannot be eliminated as \( \delta_i \) is always larger than the time unit. In a controller-generator architecture, there can be an additional outer error between the targeted feature value and the one of an actually generated segment. Hence, the overall error does not equal to inner error. An outer error, \( \varepsilon_{\text{outer}} \), and an overall error, \( \varepsilon_{\text{all}} \), are formulated in Eqs. (2) and (3), respectively.

\[
\varepsilon_{\text{outer}} = \frac{1}{T(f_{\text{max}} - f_{\text{min}})} \sum_{i=1}^{N} \delta_i |f_i - f^*_i|.
\]

\[
\varepsilon_{\text{all}} = \frac{1}{T(f_{\text{max}} - f_{\text{min}})} \sum_{i=1}^{N} \left( b_i + \delta_i \right) \sum_{t=b_i}^{b_i} |f^*_t - f_t|.
\]

The notation \( f_i \) in Eq (2) denotes the real feature value of \( S_i \). As shown in Eqs. (1), (2) and (3), those errors are normalised linearly to the range of \([0, 1]\). Fig. 2 illustrates the relationship between those errors.

Although only one level feature is considered as a case study in this work, our formulation can be easily extended to the case of using multiple level features.

B. Challenges of Music-driven Online Level Feature Control

There are at least two challenges of music-driven online level feature control, the uncertainty of play duration and the dilemma of granularity. The time consumed to play through a segment depends on the player’s skill and play style. Therefore, generating level segment from music should be achieved in a player-adaptive way. Online level generation usually generates a level segment by segment. Using segments of smaller size is expected to control features more accurately. However, online level generation also requires high generation speed. Using smaller segments will lead to a lower generation speed due to the higher frequency of making control decision. Moreover, extracting features from very small segments does not always make sense. Generating small yet reasonable segments in real-time is not trivial. To overcome the above challenges, a music-driven online level feature controller is designed and detailed in Section IV-C.

IV. ONLINE PLAYER-ADAPTIVE PROCEDURAL CONTENT GENERATION VIA REINFORCEMENT LEARNING

We propose a framework named online player-adaptive procedural content generation via reinforcement learning (OPARL) for online level generation from music. An overview of OPARL is given in Fig. 3. OPARL is composed of a feature controller and a segment generator. At each iteration, the controller receives players’ play data on the most recent segment and determines the control signal, which directs the generator to generate appropriate segments for a specific player. The generator takes a number of previous segments and the control signal as input, and then outputs a new segment. This section explains the framework of OPARL, its implementation and parameter setting used in the experimental studies. More technical details are available in the released project1.

A. Controllable EDRL with Archive

A controllable experience-driven reinforcement learning with archive (CEDRL-A) architecture, extended from the EDRL framework [11], is designed as the generator of OPARL. It models the task of online level generation as a Markov decision process so that a designer (known as “agent” in [11]) can be trained to generate rapidly level segments of high quality. A booster (known as “generator” in [11]) based on GAN is applied to tackle the high-dimensionality of designer’s action space. The booster maps a low-dimensional...
controllable way. In some primary attempts, we observe that the difficulty degree usually does not change fast in the online generation phase. Therefore, the feature is sampled uniformly in \([f_{\text{min}}, f_{\text{max}}]\), and then a Gaussian mutation is applied to the current feature value to get the next feature value recursively.

Formally, the feature values sampled during a training epoch are \(\hat{f}_i \sim U(f_{\text{min}}, f_{\text{max}})\) and \(\hat{f}_i = \hat{f}_{i-1} + \xi\) with \(\xi \sim N(0, \sigma^2)\), \(\forall i > 1\). In this work, \(\sigma\) is arbitrarily set as 0.05.

We also employ \textit{fun} and \textit{playability} introduced in the work of [11] as additional reward terms so that the generated levels are fun and playable. The playability of segment \(S_i\) is checked by simulating \(S_i + 1\) with an \(A^*\) agent that won the 2009 Mario AI Competition [28]. In our work, the \textit{playability} is set as 0 if the newly generated segment is playable, otherwise \(-1\).

2) Training Designer with Soft Actor-critic: Among the reward terms, \textit{controllability} does not depend on previous segments, and \textit{playability} depends on no more than one previous segment. However, in our setting, computing \textit{fun} requires 2 previous segments \((S_{i-1} \text{ and } S_i)\). Thus, the capacity of the generator’s archive \(m_G\) is 2. The designer uses a multi-layer perception (MLP) model. The booster uses a latent space of \(\mathbb{R}^{20}\). The targeted difficulty degree is duplicated by 12 times to increase the number of connections in the MLP on the inputted targeted difficulty. As a result, the control signal \(g_i\) has a dimensionality of 12 and the designer’s observation \(o_i\) is a 52-dimensional vector. The action space is \([-1, 1]^{20}\).

We implement a parallel-environment version of soft actor-critic (SAC) [29], [30] as an OpenAI Gym interface [31] with 50 synchronous sub-environments to train the designer for 1 million time steps in total. The actor and critic use MLP model with 3 hidden layers of 256 neurons. Our implementation of SAC updates the models 10 times using randomly sampled batches of size 384 every time 100 transitions are collected. The automating entropy adjustment [30] is used with a targeted entropy \(H_i = -20\), as recommended in [30]. The smoothing coefficient \(\gamma\) is set as 0.005. The discounted factor \(\gamma\) greatly affects the performance of reinforcement learning algorithms. Empirically, a smaller \(\gamma\) (comparing with \(\gamma = 0.99\))

\[ C(S_i) = 1 - \frac{|f(S_i) - \hat{f}_i|}{f_{\text{max}} - f_{\text{min}}} \]

In the implementation of OPARL in this paper, the difficulty degree of a level segments is considered as the feature of a level segment, quantified as the summation of the number of enemies and the number of empty ground tiles divided by the width of segment, as formulated as

\[ \text{difficulty} = \frac{\#\text{enemies} + \#\text{empty grounds}}{\text{width}}. \]

The lower bound and upper bound of difficulty degree are set as \(f_{\text{min}} = 0\) and \(f_{\text{max}} = 1\), respectively.

As the targeted feature (difficulty degree in the implementation) directly influences the next segment, the method of sampling features can affect other rewards in some unpredictable way. In some primary attempts, we observe that the difficulty degree usually does not change fast in the online generation phase. Therefore, the feature is sampled uniformly in \([f_{\text{min}}, f_{\text{max}}]\), and then a Gaussian mutation is applied to the current feature value to get the next feature value recursively. Formally, the feature values sampled during a training epoch are \(\hat{f}_i \sim U(f_{\text{min}}, f_{\text{max}})\) and \(\hat{f}_i = \hat{f}_{i-1} + \xi\) with \(\xi \sim N(0, \sigma^2)\), \(\forall i > 1\). In this work, \(\sigma\) is arbitrarily set as 0.05.

We also employ \textit{fun} and \textit{playability} introduced in the work of [11] as additional reward terms so that the generated levels are fun and playable. The playability of segment \(S_i\) is checked by simulating \(S_i + 1\) with an \(A^*\) agent that won the 2009 Mario AI Competition [28]. In our work, the \textit{playability} is set as 0 if the newly generated segment is playable, otherwise \(-1\).

2) Training Designer with Soft Actor-critic: Among the reward terms, \textit{controllability} does not depend on previous segments, and \textit{playability} depends on no more than one previous segment. However, in our setting, computing \textit{fun} requires 2 previous segments \((S_{i-1} \text{ and } S_i)\). Thus, the capacity of the generator’s archive \(m_G\) is 2. The designer uses a multi-layer perception (MLP) model. The booster uses a latent space of \(\mathbb{R}^{20}\). The targeted difficulty degree is duplicated by 12 times to increase the number of connections in the MLP on the inputted targeted difficulty. As a result, the control signal \(g_i\) has a dimensionality of 12 and the designer’s observation \(o_i\) is a 52-dimensional vector. The action space is \([-1, 1]^{20}\).

We implement a parallel-environment version of soft actor-critic (SAC) [29], [30] as an OpenAI Gym interface [31] with 50 synchronous sub-environments to train the designer for 1 million time steps in total. The actor and critic use MLP model with 3 hidden layers of 256 neurons. Our implementation of SAC updates the models 10 times using randomly sampled batches of size 384 every time 100 transitions are collected. The automating entropy adjustment [30] is used with a targeted entropy \(H_i = -20\), as recommended in [30]. The smoothing coefficient \(\gamma\) is set as 0.005. The discounted factor \(\gamma\) greatly affects the performance of reinforcement learning algorithms. Empirically, a smaller \(\gamma\) (comparing with \(\gamma = 0.99\))

\[ C(S_i) = 1 - \frac{|f(S_i) - \hat{f}_i|}{f_{\text{max}} - f_{\text{min}}} \]

In the implementation of OPARL in this paper, the difficulty degree of a level segments is considered as the feature of a level segment, quantified as the summation of the number of enemies and the number of empty ground tiles divided by the width of segment, as formulated as

\[ \text{difficulty} = \frac{\#\text{enemies} + \#\text{empty grounds}}{\text{width}}. \]

The lower bound and upper bound of difficulty degree are set as \(f_{\text{min}} = 0\) and \(f_{\text{max}} = 1\), respectively.

As the targeted feature (difficulty degree in the implementation) directly influences the next segment, the method of sampling features can affect other rewards in some unpredictable way. In some primary attempts, we observe that the difficulty degree usually does not change fast in the online generation phase. Therefore, the feature is sampled uniformly in \([f_{\text{min}}, f_{\text{max}}]\), and then a Gaussian mutation is applied to the current feature value to get the next feature value recursively. Formally, the feature values sampled during a training epoch are \(\hat{f}_i \sim U(f_{\text{min}}, f_{\text{max}})\) and \(\hat{f}_i = \hat{f}_{i-1} + \xi\) with \(\xi \sim N(0, \sigma^2)\), \(\forall i > 1\). In this work, \(\sigma\) is arbitrarily set as 0.05.

We also employ \textit{fun} and \textit{playability} introduced in the work of [11] as additional reward terms so that the generated levels are fun and playable. The playability of segment \(S_i\) is checked by simulating \(S_i + 1\) with an \(A^*\) agent that won the 2009 Mario AI Competition [28]. In our work, the \textit{playability} is set as 0 if the newly generated segment is playable, otherwise \(-1\).

2) Training Designer with Soft Actor-critic: Among the reward terms, \textit{controllability} does not depend on previous segments, and \textit{playability} depends on no more than one previous segment. However, in our setting, computing \textit{fun} requires 2 previous segments \((S_{i-1} \text{ and } S_i)\). Thus, the capacity of the generator’s archive \(m_G\) is 2. The designer uses a multi-layer perception (MLP) model. The booster uses a latent space of \(\mathbb{R}^{20}\). The targeted difficulty degree is duplicated by 12 times to increase the number of connections in the MLP on the inputted targeted difficulty. As a result, the control signal \(g_i\) has a dimensionality of 12 and the designer’s observation \(o_i\) is a 52-dimensional vector. The action space is \([-1, 1]^{20}\).

We implement a parallel-environment version of soft actor-critic (SAC) [29], [30] as an OpenAI Gym interface [31] with 50 synchronous sub-environments to train the designer for 1 million time steps in total. The actor and critic use MLP model with 3 hidden layers of 256 neurons. Our implementation of SAC updates the models 10 times using randomly sampled batches of size 384 every time 100 transitions are collected. The automating entropy adjustment [30] is used with a targeted entropy \(H_i = -20\), as recommended in [30]. The smoothing coefficient \(\gamma\) is set as 0.005. The discounted factor \(\gamma\) greatly affects the performance of reinforcement learning algorithms. Empirically, a smaller \(\gamma\) (comparing with \(\gamma = 0.99\))
in many baselines) is better. In all the experiments reported in this paper, $\gamma = 0.7$. Each epoch terminates after 50 segments are generated. For the last transition, the target Q-value is computed as $r + \gamma Q(s', \tilde{a}')$ rather than $r$ since the training aims at endless online generation.

C. Online Player-Adaptive Level Generation with Controller

Given a piece of music, our music-driven feature controller will first compute the ideal feature sequence $F^*$ during initialisation, and then store it. In addition to $F^*$, the control policy keeps an archive $X_C = \{ (\hat{f}_j, \delta_j) | j = i-m_C-1, \ldots, i-2 \}$ to keep at most $m_C$ entries of the previous segments. Each entry is composed of the targeted feature value and play duration of a segment. The last entry of $X_C$ is composed of the targeted feature value and play duration of $m_C$, and then store it. In addition to $F^*$, LS-KNN will first compute the ideal feature sequence $F^*$ and then store it. In addition to $F^*$, the control policy achieves significant performance in the simulation-based experiments detailed in Section V.

Algorithm 1 LS-KNN. In the experiments, #trial = 50, $k = 5$ and $\sigma_c = 0.02$.

| Require: $\hat{f}_{i-1}, b_i, F^*, X_C$ | Inputs |
| Require: #trials, $k$, $\sigma_c$ | Hyper-parameters |
| Ensure: $f_i$ |
| 1: $\tilde{f}_i \leftarrow \hat{f}_{i-1}$ |
| 2: $\tilde{f}_i \leftarrow \hat{f}_{i-1}$ |
| 3: $\delta_i \leftarrow \text{Estimate}(\hat{f}_i; X_C)$ |
| 4: $\tilde{\varepsilon} \leftarrow \text{Evaluate}(\hat{f}_i; \tilde{f}_i, \delta_i, F^*)$ | Least estimated error found |
| 5: repeat |
| 6: $\tilde{f}_i \leftarrow \hat{f}_i + \xi, \ \xi \sim N(0, \sigma_c^2)$ | Do local search |
| 7: $\tilde{\varepsilon}_i \leftarrow \text{Estimate}(\hat{f}_i; X_C)$ |
| 8: $\tilde{\varepsilon}_i \leftarrow \text{Evaluate}(\hat{f}_i; \tilde{f}_i, \delta_i, F^*)$ |
| 9: if $\tilde{\varepsilon}_i < \tilde{\varepsilon}$ then |
| 10: $\tilde{f}_i \leftarrow \tilde{f}_i$ |
| 11: $\tilde{\varepsilon} \leftarrow \tilde{\varepsilon}_i$ |
| 12: end if |
| 13: until has looped for #trials times |
| 14: return $\tilde{f}_i$ |

A simple algorithm named local search with KNN-based estimation (LS-KNN, Algorithm 1) is designed as the online control policy for determining targeted feature values. When determining a targeted feature value for the $i$th segment, LS-KNN executes local search for $\#trials$ generations starting from the last one $\hat{f}_{i-1}$, and picks up the best value of $\tilde{f}_i$ found according to the estimated individual inner error using KNN-based prediction of play duration for the $(i-1)$th and $i$th segments. In our case, an individual is mutated by adding a Gaussian noise with a standard deviation of $\sigma_c$. The play duration is estimated with:

$$\text{Estimate}(\hat{f}_i; X_C) = \frac{1}{k} \sum_{j \in J} \delta_j,$$

where $J$ is the set of $k$ nearest neighbours in terms of $|\hat{f}_j - \tilde{f}_i|$ within $X_C$. With $\text{Estimate}(\hat{f}; X_C)$, we can further estimate the time that the $i$th segment $\tilde{f}_i$ starts to be played. With $\delta_j$, the estimated play duration respected to $\tilde{f}_i$, a feature can be evaluated using:

$$\text{Evaluate}(\hat{f}_i; \delta_j, \tilde{f}_i, F^*) = \frac{1}{\delta_i} \sum_{t=b}^{\tilde{b}+\delta_i} |f_t^* - \tilde{f}_i|.$$

Algorithm 1 details the implementation of the control policy. LS-KNN is proposed with two assumptions: (i) an ideal feature sequence won’t change fast, thus the “optimal” $\hat{f}_i$ should not be far from $\hat{f}_{i-1}$; (ii) the difficulty implies the play duration, which is true for many platformer games. The former is somehow generic in online level generation because fast changes of features may be harmful to the coherence of levels. It is the reason of starting the local search from $\hat{f}_{i-1}$ in Algorithm 1. The latter motivates the estimation of play duration according to the records organised by $(\hat{f}_i, \delta_j)$.

LS-KNN is easy to implement and can always be used directly without training or other preparation. Furthermore, though not explicit, LS-KNN is well player-adaptive since the KNN-based estimation is applied based on the specific player’s play data collected online. Our implementation with LS-KNN policy achieves significant performance in the simulation-based experiments detailed in Section V.

D. Implementation of Booster

Our booster is a variation of MarioGAN [15]. Differ from the original neural network architecture, our GAN model uses fractional-convolutional layers [32] with kernel size, stride and padding of $\langle (4, 4), (1, 1), 0 \rangle$, $\langle (3, 3), (2, 2), 1 \rangle$, $\langle (4, 4), (2, 2), 1 \rangle$ and $\langle (3, 4), (1, 2), 1 \rangle$, respectively, to directly obtain an output of size $14 \times 28$ without clipping. Besides, latent vectors of length 20 are used. The generator and discriminator are trained for 5 times and 1 time at each iteration, respectively, on the 13 human-designed levels without bullet bills provided in the Video Game Level Corpus [33].

V. EXPERIMENTAL STUDY

To evaluate the effectiveness of our approach and implemented algorithms in optimising different objectives, the designer is trained with all the possible combinations of the three reward terms presented in Section IV-B1 with the same weights, and evaluated with training environments and online generation environments, respectively.

To test the robustness of our method, five different agents in the 2009 Mario AI Competition [28], namely Baumgarten’s (the aforementioned $A^*$ agent), Sloane’s, Hartmann’s, Polikarpov’s and Schumann’s agents, are used as the simulated player, and two different pieces of music, Ginseng3 (EnV, 2014) and Farewell4 (Raine, 2019), are used. Fig. 4, shows the five agents’ play duration on each segments of an level generated online by OPARL using a designer trained with the summation of controllability, fun and playability. Those agents

3From the original sound track of commercial platformer game Electronic Super Joy: Groove City (Michael Todd, 2014).
4From the original sound track of commercial platformer game Celeste (Matt Makes Games Inc., 2018).
Table I shows the experimental results. The designers trained with controllability generally achieve very low overall error in the online generation tests. The main source of overall error is the outer error, i.e., the error between the targeted feature value produced by controller and the feature of actually generated segment. The value of $1 - C$ closed to $\varepsilon_{\text{outer}}$ means that our method of sampling targeted features is effective. The designer trained with only fun reward achieves a great score on $\sqrt{-F}$. However, when controllability is employed, the score of fun deteriorates a lot. This phenomenon indicates that the objective of fun and controllability conflict. Moreover, the controllability deteriorates less comparing with the designer trained with controllability only. It is probably because the reward of fun uses a quadratic form while controllability uses a linear form. That means designer finds it better to optimise controllability to get a higher summation of reward terms.

All the designers trained with playability well ensure the playability of generated levels, while designers generally assure better playability with the help of resampling. A merit attention finding is that the designer trained with $F$ and the designer trained with $C + F + P$ do not get notable better $P$ value in the online generation tests. A possible reason is that those designers lack of randomness when taking actions. That means if they generate an unplayable segment, no matter how many times the re-generation is executed, they will still generate unplayable segments. A future work is finding out why the phenomenon only appears on those two designers.

As a conclusion, our implemented framework optimises the reward functions effectively. The designer trained with $C + F + P$ balances different objectives and can be a good choice for online level generation from music.

B. Evaluation of Robustness

Fig. 6 plots the overall error, fun and diversity evaluated on the designer trained with $C + F + P$ as reward for the five agents and two different pieces of music. The overall error and fun are plotted as $1 - \varepsilon_{\text{all}}$ and $1 - \sqrt{-F}$ for better intelligibility. According to Figs. 6(a) and 6(b), our method achieves very similar and high performances of overall error and fun, and is robust for players with different play speed. The diversity scores of levels generated for different players and musics vary significantly. According to Figs. 4 and 5, the diversity of levels generated by OPARL may be positively correlated with the fluctuation degree of music and the variance of play duration.

Fig. 7 presents segments captured from the levels generated with the same $S_0$ (i.e., initial segment) for different agents and different music. Fig. 1 uses an illustration to explain how OPARL generates different levels for different players. It is shown in Fig. 7 that levels generated from Ginseng are generally harder than the ones generated from Farewell, as the ideal difficulty sequence derived from Ginseng is generally larger than the one from Farewell (cf. Fig. 5). The levels generated for different agents with the same music are similar. It may be explained by using the same starting segment to

Table I shows the experimental results. The designers trained with controllability generally achieve very low overall error in the online generation tests. The main source of overall error is the outer error, i.e., the error between the targeted feature value produced by controller and the feature of actually generated segment. The value of $1 - C$ closed to $\varepsilon_{\text{outer}}$ means that our method of sampling targeted features is effective. The designer trained with only fun reward achieves a great score on $\sqrt{-F}$. However, when controllability is employed, the score of fun deteriorates a lot. This phenomenon indicates that the objective of fun and controllability conflict. Moreover, the controllability deteriorates less comparing with the designer trained with controllability only. It is probably because the reward of fun uses a quadratic form while controllability uses a linear form. That means designer finds it better to optimise controllability to get a higher summation of reward terms.

All the designers trained with playability well ensure the playability of generated levels, while designers generally assure better playability with the help of resampling. A merit attention finding is that the designer trained with $F$ and the designer trained with $C + F + P$ do not get notable better $P$ value in the online generation tests. A possible reason is that those designers lack of randomness when taking actions. That means if they generate an unplayable segment, no matter how many times the re-generation is executed, they will still generate unplayable segments. A future work is finding out why the phenomenon only appears on those two designers.

As a conclusion, our implemented framework optimises the reward functions effectively. The designer trained with $C + F + P$ balances different objectives and can be a good choice for online level generation from music.

B. Evaluation of Robustness

Fig. 6 plots the overall error, fun and diversity evaluated on the designer trained with $C + F + P$ as reward for the five agents and two different pieces of music. The overall error and fun are plotted as $1 - \varepsilon_{\text{all}}$ and $1 - \sqrt{-F}$ for better intelligibility. According to Figs. 6(a) and 6(b), our method achieves very similar and high performances of overall error and fun, and is robust for players with different play speed. The diversity scores of levels generated for different players and musics vary significantly. According to Figs. 4 and 5, the diversity of levels generated by OPARL may be positively correlated with the fluctuation degree of music and the variance of play duration.

Fig. 7 presents segments captured from the levels generated with the same $S_0$ (i.e., initial segment) for different agents and different music. Fig. 1 uses an illustration to explain how OPARL generates different levels for different players. It is shown in Fig. 7 that levels generated from Ginseng are generally harder than the ones generated from Farewell, as the ideal difficulty sequence derived from Ginseng is generally larger than the one from Farewell (cf. Fig. 5). The levels generated for different agents with the same music are similar. It may be explained by using the same starting segment to

Table I shows the experimental results. The designers trained with controllability generally achieve very low overall error in the online generation tests. The main source of overall error is the outer error, i.e., the error between the targeted feature value produced by controller and the feature of actually generated segment. The value of $1 - C$ closed to $\varepsilon_{\text{outer}}$ means that our method of sampling targeted features is effective. The designer trained with only fun reward achieves a great score on $\sqrt{-F}$. However, when controllability is employed, the score of fun deteriorates a lot. This phenomenon indicates that the objective of fun and controllability conflict. Moreover, the controllability deteriorates less comparing with the designer trained with controllability only. It is probably because the reward of fun uses a quadratic form while controllability uses a linear form. That means designer finds it better to optimise controllability to get a higher summation of reward terms.

All the designers trained with playability well ensure the playability of generated levels, while designers generally assure better playability with the help of resampling. A merit attention finding is that the designer trained with $F$ and the designer trained with $C + F + P$ do not get notable better $P$ value in the online generation tests. A possible reason is that those designers lack of randomness when taking actions. That means if they generate an unplayable segment, no matter how many times the re-generation is executed, they will still generate unplayable segments. A future work is finding out why the phenomenon only appears on those two designers.

As a conclusion, our implemented framework optimises the reward functions effectively. The designer trained with $C + F + P$ balances different objectives and can be a good choice for online level generation from music.

B. Evaluation of Robustness

Fig. 6 plots the overall error, fun and diversity evaluated on the designer trained with $C + F + P$ as reward for the five agents and two different pieces of music. The overall error and fun are plotted as $1 - \varepsilon_{\text{all}}$ and $1 - \sqrt{-F}$ for better intelligibility. According to Figs. 6(a) and 6(b), our method achieves very similar and high performances of overall error and fun, and is robust for players with different play speed. The diversity scores of levels generated for different players and musics vary significantly. According to Figs. 4 and 5, the diversity of levels generated by OPARL may be positively correlated with the fluctuation degree of music and the variance of play duration.

Fig. 7 presents segments captured from the levels generated with the same $S_0$ (i.e., initial segment) for different agents and different music. Fig. 1 uses an illustration to explain how OPARL generates different levels for different players. It is shown in Fig. 7 that levels generated from Ginseng are generally harder than the ones generated from Farewell, as the ideal difficulty sequence derived from Ginseng is generally larger than the one from Farewell (cf. Fig. 5). The levels generated for different agents with the same music are similar. It may be explained by using the same starting segment to
TABLE I: Evaluation of designers trained with different reward functions. All values are averaged over 100 independent trials. \( \text{Div} \) is to be maximised, all the other metrics are to be minimised with strict lower bound of 0. Cells filled with \(-/-\) are meaningless. The best and worst results are highlighted with bold and italic, respectively.

<table>
<thead>
<tr>
<th>Designer</th>
<th>Training Environment</th>
<th>Online Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \sqrt{F(10^{-2})} )</td>
<td>( P(10^{-2}) )</td>
</tr>
<tr>
<td>( F )</td>
<td>1.08 ± 0.34</td>
<td>9.12 ± 4.03</td>
</tr>
<tr>
<td>( P )</td>
<td>11.2 ± 1.81</td>
<td>0.22 ± 0.63</td>
</tr>
<tr>
<td>( F+P )</td>
<td>2.06 ± 0.58</td>
<td>0.42 ± 0.82</td>
</tr>
<tr>
<td>( C )</td>
<td>16.8 ± 3.27</td>
<td>46.7 ± 21.2</td>
</tr>
<tr>
<td>( C+F )</td>
<td>4.48 ± 2.13</td>
<td>15.3 ± 6.76</td>
</tr>
<tr>
<td>( C+P )</td>
<td>11.9 ± 3.41</td>
<td>0.40 ± 0.85</td>
</tr>
<tr>
<td>( C+F+P )</td>
<td>6.81 ± 1.31</td>
<td>0.40 ± 0.94</td>
</tr>
</tbody>
</table>

Fig. 6: Values of \( \text{fun}, \text{controllability}, \text{diversity} \) evaluated on designers trained with different agents as player and different musics. Each value is averaged over 30 independent trials.

generate those levels. To summarise, our method can adapt well different players and is robust to different music.

VI. CONCLUSION

This paper formulates the problem of online level generation from music, and proposes an online player-adaptive procedural content generation (OPARL) framework composed of a novel CEDRL-A generator and a novel LS-KNN controller to achieve online level generation from music. Experimental results show that the implementation of OPARL can generate in real-time SMB levels with segment-wise features closed to an ideal difficulty sequence derived from a piece of music. The resulted generation system can also guarantee the playability. The training algorithm implemented in this paper achieves considerable performance and can be used as a baseline in further studies. Moreover, our framework is flexible since the controller and the generator are decoupled. The CEDRL-A generator in our framework can be integrated with other controllers like DDA controller for different aspects of player-adaptation.

In this paper, our proposed approaches are verified with simulation-based studies. One of the future work is conducting human tests. As another future work, new ways of mapping multiple features of both levels and music can be studied for the purpose of achieving better consensus between play experience and music.

ACKNOWLEDGEMENT

The authors would like to thank the anonymous reviewers for their valuable comments.

REFERENCES

Fig. 7: Levels generated from an identical initial segment using the designer trained with $F+C+P$ as reward, different musics and different agents as player. Levels are captured from the 21st segments, as the controller's archive is fulfilled after 20 segments were generated.


