Scientia Iranica D (2021) 28(3), 1497–1514

Sharif University of Technology
Scientia Iranica
Transactions D: Computer Science & Engineering and Electrical Engineering
http://scientiairanica.sharif.edu

A dynamic balanced level generator for video games based on deep convolutional generative adversarial networks

M. Rajabi, M. Ashtiani*, B. Minaei-Bidgoli, and O. Davoodi

Computer Games Research Laboratory, School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran.

Received 31 October 2019; received in revised form 20 August 2020; accepted 16 November 2020

KEYWORDS
Generative adversarial networks;
Dynamic difficulty adjustment;
Reinforcement learning;
Video games;
Game balance.

Abstract. In the gaming industry, creating well-balanced games is one of the major challenges developers are currently facing. Balance in games has different meanings depending on the game type. But, most existing definitions are defined from flow theory. In this research, Generative Adversarial Networks (GANs) have been used to automatically create balanced levels. In the proposed work, a level of a 2D platformer game is considered as a picture and is fed to the network. The levels are randomly created while adhering to a set of balance requirements. Those levels that can be solved with the help of an agent using reinforcement learning in the number of tries set by designers are given as input data to the network. Finally, the network automatically generates new balanced levels and then, the levels are checked to see if they have the game's minimum necessary requirements and also to check if they can be solved by the reinforcement learning agent. The best performing network is then selected for the level generation. In the series of performed evaluations, it is shown that after the training process, the proposed approach is capable of generating levels that are well-balanced with considerable accuracy.

© 2021 Sharif University of Technology. All rights reserved.

1. Introduction

In the game development process and, especially, game design, balance is one of the most difficult and time-consuming activities [1]. As Ernest Adams in his well-known book 'Fundamentals of game design' has stated: "To be enjoyable, a game must be balanced well–it must be neither too easy nor too hard, and it must feel fair, both to players competing against each other and to the individual player on his own." [2] This definition is inspired by the flow theory introduced by Mihaly Csikszentmihalyi, a famous psychologist [3]. He hypothesized that a person's skill and the difficulty of a task interact with a result of cognitive and emotional states. Due to the diversity of video game styles, balance in games is defined in different ways. For example, in turn-based games, designers tend to balance the game in order to make fairness in the win-rate [4] or, in other games, designers tend to adjust the difficulty of passing levels and balance them dynamically [5–7]. In strategy games, designers try to find the unbalanced behaviors of the players to prevent the formation of dominant winning strategies [8]. In general, and as a widely used definition, the meaning of balance in a video game is that playing the game and passing the levels by the player does not go beyond the framework of what the game designer has imagined or intended to happen [9,10].

Recently, a large amount of research has been

*. *Corresponding author. Tel/Fax: +98 21 73021480*
E-mail addresses: morteza_rajabi@alumni.iust.ac.ir (M. Rajabi); m_ashtiani@iust.ac.ir (M. Ashtiani); b_minaei@iust.ac.ir (B. Minaei-Bidgoli); omiddavoudi@cmail.carleton.ca (O. Davoodi)

doi: 10.24200/sci.2020.54747.3897

performed on creating balance in computer games. Bosc et al. [8], Karavolos et al. [11], Olesen et al. [12], Morosan and Poli [13], Bangay and Makin [14] and Uriarte and Ontanón [15] have all tried to address this issue using different approaches. Research based on manual human-based decision making is more accurate compared to automatic balanced level generators. However, this system being time-consuming and error-prone has encouraged researchers to explore other methods in which an automated intelligent algorithm plays an essential role. These methods are often faster than the manual level creation process in detecting and fixing bugs in the created levels. But, mistakes may still occur when the human expert wants to choose the proper parameters and settings for the level generation algorithm. Thus, in the end, human-error may manifest itself in another form, impacting the balance of the created level. In this paper, a method is proposed where the power of recently introduced deep convolutional Generative Adversarial Networks (GAN) is leveraged [16]. Currently, deep convolutional GANs are gaining significant success in the field of image processing. The general similarities and the potential mapping between these two domains was a motivation to use these networks for the automatic generation of balanced game levels.

To this aim, a method for balancing video games using deep convolutional GANs is proposed, in which the map of the game, as well as the corresponding parameters, will be automatically determined without any human intervention. The way these networks work in the field of image processing is that they are fed a number of images as input. Then, after the training steps are completed, they generate new images (i.e. not existing in the training input set) that are similar to the input images. Of course, GANs have found many other useful application domains as well. For example, GANs have been successfully applied in handwritten digit generation [17], human face creation [18], and even creating bedroom layouts [19]. GANs have also been employed in the field of video and sound analysis such as frame prediction in videos [20,21] and sound generation [22].

To obtain the initial data, a basic level generator was implemented that randomly generated the levels without considering whether the level would be solvable or not. To effectively use GAN, one needs a huge amount of input data. If a human expert was assigned to determine whether the game levels are solvable or not in a manual process, it would take a very long time to do so, making the approach almost impractical. Therefore, reinforcement learning approaches have been used [23,24] to simulate the human play process. Then, after performing the repetitions, a large number of solvable levels are generated and given as input to the GAN. Subsequently, the network is prepared to build levels that have well-known features by specifying and changing various parameters. In summary, the following are the major contributions of the proposed approach:

1. Providing a metric for evaluating different procedural level generation approaches and especially GAN-based approaches. To the best of the authors' knowledge, there is no such metric and evaluation mechanism in the literature yet and, usually, the evaluation is performed by a human operator using a manual evaluation process.
2. Introducing an approach for random balanced level generation using deep convolutional GANs. The solvability of the level is determined using a reinforcement learning agent allowing the level designer to define the requirements for a balanced level and automatically generate such levels without any human involvement.
3. Leveraging the power of reinforcement learning approaches to determine the level difficulty and the adherence of the created levels to the balance requirements specified by the level designer.

In the remainder of this paper and in Section 2, the related work of this research will be reviewed. In Section 3, the background knowledge required for the proposed approach is given. In Section 4, the proposed model for automatic balanced level generation using GANs will be introduced. In Section 5, the results of the evaluation of the proposed method are given. Finally, the paper concludes in Section 6.

2. Related work

A variety of research has been performed in past years on automatic game balancing and dynamic difficulty adjustment. In this section, the related work of the research in this paper will be reviewed and their strengths and weaknesses analyzed.

In one of the first attempts to use GANs in the context of game level generation, Volz et al. used the Mario game levels [25] to investigate the possibility of using GANs for generating new levels [26]. For doing this, the authors created pre-designed levels, formatted them in 28*14 windows and fed them to the GAN. Then, the generative network will output the new levels. After the creation of the new levels, the authors have used an A^* agent used in the Mario AI Competition in 2009 to determine the solvability of the levels. In this research, latent variable evolution [27] is used to investigate how latent GAN vectors can be evolved through a fitness-based approach in the context of level generation. Also, a CMA-ES [28] strategy is used to evolve the latent vector.

In another work, Shaker et al., aggregated many recent approaches to procedural content generation in

computer games [29]. Different approaches such as search-based techniques [30], L-systems [31], constructive generation [32] and Fractal methods [33] are investigated in a wide range of game genres, such as card-games and 2D platformer games. Besides the numerous advantages that each of these approaches provides, a common downside is the role of human operators to hard-code the knowledge of content creation for every specific context. This is indeed not the case in the approach taken in this paper, where domain knowledge is created using a reinforcement learning approach.

Similarly, Pérez et al. researched the mechanisms to create balanced AI in video games [5]. The challenge they are tackling is to create a compelling experience for the professional, as well as novice, players of a 2D platformer game. Using a compelling experience, the authors illustrate that the new and novice players should not leave the game because of its high difficulty and the professional players because of its simplicity. The method they have used in their study is to dynamically change the parameters of the game such as player speed, number of obstacles and so on. These changes will dynamically occur according to how the player plays the game. In this study, a runner 2D platformer game was designed. Then, different game parameters such as speed, type, and cycle of obstacles were linked together with the help of evolutionary Fuzzy Cognitive Maps (FCM) [34,35]. In this research, the parameters are manually linked together by the authors in a predetermined way. The simplicity and soundness of the proposed approach aside, the extraction of influential parameters in the level and determining their impact on the difficulty of the game is a challenging task for the level designer and is prone to much human error. In addition, creating the predetermined influence factors and setting their correct coefficient degree is not a simple job and requires an extensive amount of trial and error by the human operator.

In another interesting work, Morosan et al. used the class of evolutionary algorithms [36,37] in order to find the values of the parameters leading to game balance [38]. This algorithm is a subclass of the genetic algorithm [39]. In this work, by defining the fitness function [40] (a function of player's winning rate and the amount of difference between the new parameters and the old), the parameters of the game will dynamically change to reach the balance the game designer has desired. In this research, the game designer first chooses the expected winning rate of the level. The Pacman game is tested as a sample. The game was simulated for a large number of repetitions using evolutionary genetic algorithms. The winning rate value of the new parameters compared with the initial parameters was recorded. According to the balance function formula, the experimental values that had a smaller difference with the initial values, as well as a winning rate that was nearly expected, are selected as the new balanced parameters. After the successful results in the Pacman game, these experiments were also examined in the Star Wars game and achieved similarly satisfying results. The use of the genetic algorithm in this study causes balanced values to be as close as possible to the values that the designer expects. However, the selection of parameters by humans may cause many errors. For example, a parameter that does not have a significant effect on the balance of the game may mistakenly affect the evolution process and create a significant impact on the accuracy of the newly modified parameters.

Xia et al. have also researched the area of game balance using the procedural content generation technique [41] (A similar approach is also taken by [42]). They have used this method on a game designed with the Unity3D engine to evaluate their research. The game is a two-dimensional shooting game, in which the player is attacked by several enemy types. When the player starts the game, their Mana is 0. Therefore, the Mana left to the player at the end of each turn (which can be positive or negative) is considered as an offset value. The aim of this research is to find a way wherein at the end of each turn, the player ends up with zero Mana points. This is performed with the help of the PSO [43] and dynamic behavior-changing [34] techniques. Although the proposed approach is successful in creating balance in a well-structured way, the approach is time-consuming to implement and learn.

In another research line, Silva et al. investigated the dynamic difficulty adjustment based on the players' behavior type [6]. The DOTA game has been used for this research. As the first step, a general gameplay pattern of different players is extracted. From this pattern, it is pointed out that the complexity of the game for novice players causes them to leave the game. Also, the simplicity of the game for professional players makes them get bored quickly. Therefore, the authors have tried to set up a strategy that adjusts the difficulty level of the game according to each individual player. For example, if the player is a beginner, the difficulty level of AI would decrease to some extent so that the player would be either winning or getting defeated in a balanced manner. To evaluate the performance of this research, a number of features have been extracted from the game and then a relationship for evaluation has been introduced. It is shown that the approach can flexibly change the difficulty level of the game based on the players' behavior. Of course, creating multiple AI scenarios and triggering each of them based on a certain condition requires extensive and accurate AI design.

In the very limited research where deep neural networks have been used, Karavolos et al. have used this method to balance the levels and parameters of

a First-Person Shooter (FPS) game [11]. Adjustable parameters in this study are gameplay maps as well as the game's weapon parameters [44,45]. The idea behind this research is to use deep neural networks to detect balanced levels. For doing so, the game designer determines the map of a level along with the parameters of the weapons. Then, the neural network will comment on whether it is balanced or not. The strength of this study is that, after the learning stage and at a high speed, the network recognizes the designed levels and tags them as balanced or unbalanced. But, it still requires a human operator at several stages in the process (i.e. feature extraction, network parameter design and so on).

From the very little research that has applied basic GAN as its method of level generation, Giacomello et al. used such networks to generate procedural content [46]. The required data for the neural network are obtained through human-made levels and stored as a WAD file. WAD format files include all the level's information. In order to evaluate the results, the authors used the SLAM algorithm [47]. But, by reviewing the body of research performed in the field of GANs, it can easily be inferred that most of these studies are conducted with a much larger number of input data compared to this research. So, it seems that the volume of data in this study, due to the size of the images and the number of features, is forcibly limited to a very low amount.

In another research, Morosan and Poli investigated balancing in the PacMan game [13]. Designers have a set of goals and rules in their minds that they wish to observe in the game. Because of that, a useful tool for game designers has been presented for use, in order to turn the designers' vision of a game into reality as accurately and easily as possible. Neural networks and genetic algorithms have been used in order to generate agents of various skills and this is particularly valuable for cases where no pre-existing agent exists.

Bangay and Makin undertook research regarding achieving balance in a large real-time strategy game [14]. They proposed an attribute space representation as a common framework for reasoning about balance in the combat scenarios found in these games. The typical attributes of range, speed, health, and damage have been used in this work. For measuring balance in this type of game, authors have paid attention to the health of each unit at the end of the game and have calculated the difference between each unit's health (h_1, h_2). If $h_1 - h_2$ is positive, it means that team 1 wins, if it is negative, it means that team 2 wins, and if it is 0, it means the game is balanced. The proposed model can predict the win probability and it helps to improve the game balance. This framework is a suitable tool for identifying the unbalanced situation in the RTS game, but it is created only for RTS games and the definition of balance in this work is too limited (if the difference between unit health becomes zero, it means that game is balanced).

Finally, Uriarte and Ontanón have also investigated generating balanced maps for StarCraft [15]. PSMAGE has been presented in order to generate balanced maps for this popular real-time strategy (RTS) game. This approach has used Voronoi diagrams to generate an initial map layout and after that, different properties were assigned to each of the regions in the diagram. This approach generates the map procedurally. After generating maps, PSMAGE uses a collection of evaluation metrics, in order to measure how balanced a map is. Balanced Map in this work refers to maps having two conditions. First, if all the players have the same skill level, they should have the same chances of winning the game at the end. Second, in the case of StarCraft, no race should have a significant advantage over the others. In Table 1, the approaches are summarized and compared.

3. Background knowledge

This section reviews the basic concepts of GANs and reinforcement learning algorithms. In our proposed approach, GANs have been used to create new levels, using data derived from reinforcement learning and simulation methods. In this section, these two approaches are briefly overviewed.

3.1. Generative adversarial networks

Generative adversarial networks were first introduced by Goodfellow et al. in 2014 [16]. These networks are considered to be a subclass of deep learning algorithms. In fact, they are a neural network composed of two main components, namely, the generator and the discriminator:

1. **Generator:** The generator is fed real numbers as input. The output of the generator which is equal to the size of the input dataset is the data that the generator tries to generate with the characteristics of the input dataset;
2. **Discriminator:** The task of the discriminator is to identify fake data. The input of this network is the data from the real data set (marked with a real tag), as well as the output of the generator (marked with a fake tag). The purpose of the discriminator is to recognize these labels appropriately and separate real and fake data.

These two components complement one another to build the data correctly and with the properties of the input data set. The function of this neural network is as follows: First, the generator produces a given number of random noise data as input and sends them as output to the discriminator using the forward

Table 1. A comparison of game balance approaches.

	Ability to change the game mechanism and parameters	**Ability to generate levels**	**Ability to check the balance in levels**	**Dynamic difficulty adjustment**	**Using RL in order to check level solvability and difficulty state**	**Using GAN for generating levels**	**Not using human-generated levels as a dataset**	**Allowing designers to define balance in a customized way**	**Ability to potentially use the proposed model in other game genres**
Proposed approach	√	√	√		√	√	√	√	√
Evolving Mario levels in the latent space of a deep convolutional generative adversarial network		√	√			√			√
Dynamic game difficulty balancing in real-time using evolutionary fuzzy cognitive maps	√			√			√		
Automated game balancing in Ms. PacMan and StarCraft using evolutionary algorithms	√						√	√	√
Game balancing with ecosystem mechanism			√				√		√
Dynamic difficulty adjustment through an adaptive AI				√			√	√	
Learning the patterns of balance in a multi-player shooter game			√				√		√
DOOM level generation using generative adversarial networks		√	√			√			√
Balancing players of different skill levels in a first-person shooter game	√			√			√	√	√
An automated high-level saliency predictor for smart game balancing	√	√	√				√	√	
Real-time challenge balance in an RTS game using rtNEAT				√			√		√
Evolving a Designer-Balanced Neural Network for Ms PacMan	√			√			√		
PSMAGE: Balanced map generation for StarCraft		√	√						√
Generating an attribute space for analyzing balance in single unit RTS game combat			√				√		
Dynamic Difficulty adjustment of Game AI for video game dead-end				√			√		√
Balancing turn-based games with chained strategy generation			√				√		√

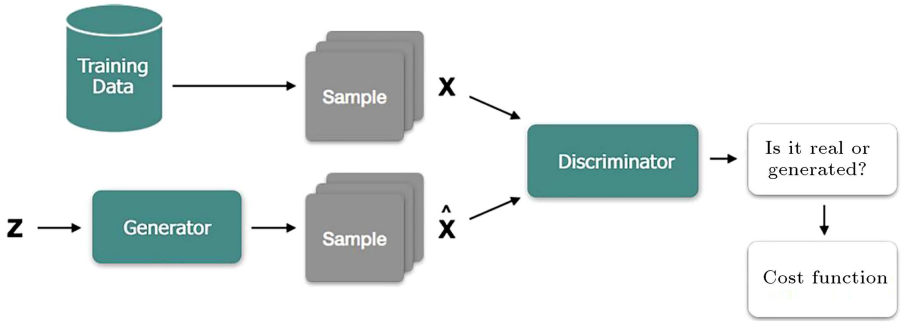

Figure 1. GAN's architecture [48].

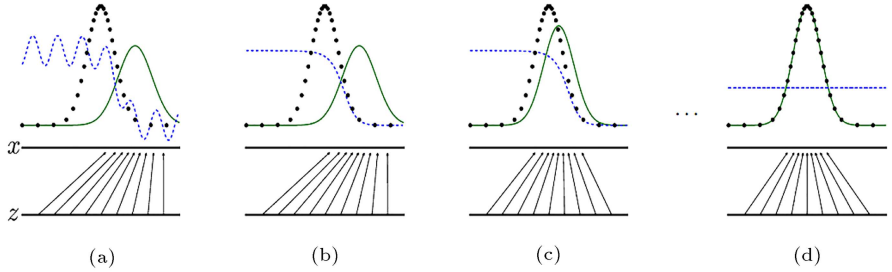

Figure 2. The convergence of data made by the generator toward the input data [16].

propagation technique. The discriminator receives the generator's data and with forward propagation, calculates the network's output. The generator keeps the weights fixed while calculating the error and, using the backward propagation technique, the weights of the discriminator are modified with the aim of better diagnosing the real and fake data. Then, the generator produces a number of new noise data and with forward propagation, the output will be calculated. As the generator is trying to trick the discriminator, this new output is labeled as genuine. In this step, the discriminator's weights are kept fixed and, by calculating the error and using the backward propagation technique, the generator's weights are updated.

In Figure 1, the architecture of the GAN is shown. The parameter z is the random input vector given to the generator. The parameter X is the training data that is produced by the generator from the random input vector. These are fed to the discriminator so that it can distinguish between real and fake constructed data. Then, the error rate is calculated using the cost function which will affect the weights of the generator and discriminator. If we call the data produced by the generator as p_g, then the generator's purpose is that p_g should have the features of X as the input data. If we represent the random vector with z, then $G(z;\theta_g)$ is the function that converts the random vector into the input data space. The parameter θ_g in the above function represents the parameters of the multi-layer perceptron network. The discriminator is denoted by $G(x;\theta_d)$, where X represents the input data. The output of this function is a value indicating whether this data is real or artificially constructed by the generator. Similar to the generator function, θ_d denotes the discriminator's parameters of the multi-layer perceptron network. Simply, the GAN acts like a min-max dual game.

If this game is represented by $V(G,D)$, the GAN operates using Eq. (1):

$$\min_G \max_D V(D,G) = E_{x\sim P_{data}(x)}[\log D(x)] + E_{Z\sim P_Z(z)}[\log(1-D(g(z))], \quad (1)$$

where G is the generator network, D is the discriminator network, E is the cost function, and $G(z)$ is the output of the generator network.

If a sufficient amount of time and input data are available, the general answer to the above equation would be that the data artificially produced by the generator will be exactly equal to the input data (i.e. $p_g = p_{data}$). This is shown in Figure 2. As can be seen in the figure, the data produced by the generator is converged over time with the input data. In this figure, the bottom solid line denoted by z displays the random vector domain and the upper solid line denoted by x displays the input domain. Vertical lines are drawn from z to x to show how to map a random vector domain to a data domain. Step (a) is the beginning of the network's training, whereas steps (b), (c) and (d), respectively, show how the generated data is converged toward the input data.

Many studies have been performed in the domain of image processing using GANs. In Figure 3, the

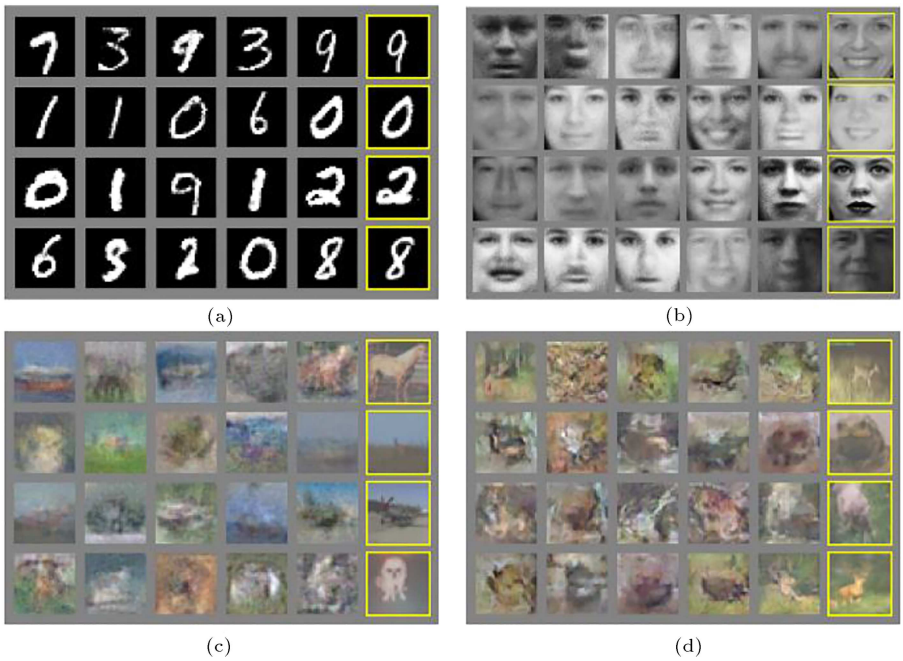

Figure 3. Outputs of GAN compared to the original data in (a) MNIST dataset, (b) TFD dataset, (c) CIFAR-10 dataset (fully connected model), and (d) CIFAR-10 dataset (convolutional discriminator and deconvolutional generator [16]).

results obtained from using a GAN are shown in four different datasets. Figure 3(a) shows how a GAN manages to generate handwritten numbers. Figure 3(b) illustrates the output of a GAN in the domain of human face generation. In Figure 3(c) and (d), the GAN is used to produce animal images. The difference in the output of a GAN depends on the network architecture as well as the input data collection mechanism, as shown in the figure.

3.2. Reinforcement learning

Reinforcement learning is a class of machine learning algorithms where an agent learns optimal behavior by interacting with the environment and getting rewards for its actions. In each step, the agent perceives the environment and chooses the action it considers the best. By implementing the action, the state of the environment might change and the agent is given a numerical reward. The ultimate goal of the agent is to maximize the gained reward over time. Figure 4, shows the general scheme of a reinforcement learning algorithm.

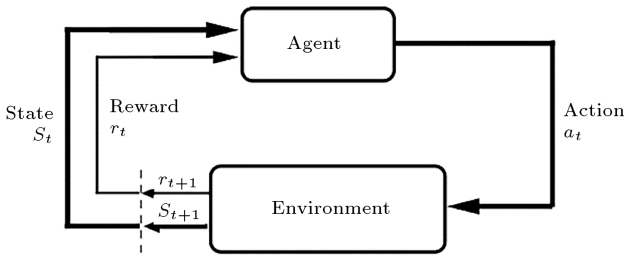

Figure 4. The general schema of the reinforcement learning agent [23].

Reinforcement learning is usually modeled using a Markov decision process [49]. A Markov decision process can be defined as (S, A, t, r), where S is a finite set of possible environment states, A is a finite set of actions, $t : s \times A \times S \rightarrow [0, 1]$ is a function detailing the probability that the state of the environment changes if the agent implements an action, and $r : s \times A \times S \rightarrow R$ is the numerical reward the agent gains by implementing an action resulting in a particular state transition.

3.3. Q-learning

One of the most popular temporal difference reinforcement learning methods is Q-learning [50]. In its basic form, it can be written as Eq. (2):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + y\max_a Q(S_{t+1}, a) - Q(S_t, A_t)], \quad (2)$$

where S is the environment state, A is an action, t is the reward, and index S is the current time step. Also, $Q(S, A)$ is a function of how valuable it is to implement action A when the state of the environment is S. The equation offers a way to update the Q function when the agent implements action A_t in environment state S_t, which results in the reward R_{t+1} and a new state S_{t+1}. As the agent interacts with the environment, the Q function is updated until the gained reward for any episode is maximized. As a result, the Q-learning algorithm can be written as Algorithm 1, as shown in Figure 5.

```
Input:
 (1) K: number of training episodes
 (2) M: number of steps for each episode

 PROCEDURE:
1:   Initialize Q(S,A), ∀s ∈ A(s) arbitrarily, and Q(terminal state) = 0
2:   FOR (i = 0; i < K; i++)
3:       Initialize S
4:       FOR (j = 0; j < M; j++)
5:           Choose A from S using policy derived from Q (e.g. ε-greedy).
6:           Take action A, observe R and S'.
7:             Q(S_t,A_t) ← Q(S_t,A_t) + α[R_{t+1} + y max_a Q(S_{t+1},a) − Q(S_t,A_t)]
8:             S ← S'
9:           IF (S is terminal)
10:              break
11:          END-IF
12:      END-FOR
13:  END-FOR
 END-PROCEDURE
```

Figure 5. Algorithm 1: Q-learning: An off-policy TD control algorithm.

3.4. Deep Q-networks

Deep-Q-Networks, or DQNs, try to model the Q function with a deep neural network, usually a convolutional neural network, so that the output is a vector showing the Q values for each possible action of the agent. Eq. (3) shows the way DQN is updated:

$$Y_t^{DQN} \equiv R_{t+1} + y \max_a Q(S_{t+1}, a; \theta_t^-). \tag{3}$$

DQN is inherently unstable and uses a mechanism called experience replay to use older experience for learning the Q value. This decreases the instability of the network and helps to improve the learning process dynamically.

3.5. Hierarchical reinforcement learning

One of the problems of reinforcement learning is the curse of dimensionality, where the increase in the dimensions of the problem exponentially increases the time needed for the agent to learn the optimal policy. One of the ways to tackle this problem is to define abstraction layers in the form of temporal or spatial abstractions to reduce the dimensionality of the problem and improve the learning speeds. This type of reinforcement learning is called hierarchical reinforcement learning [51].

3.5.1. The options framework

One of the most popular frameworks proposed for hierarchical reinforcement learning is the options framework [51]. In this framework, a number of options, which are macro actions that consist of a number of low-level actions, are defined. Each option O can be defined as $< I, \pi, \beta$, where I is a set of environment states where the option can be called, π is the policy for this particular option, and β is the probability where the option is terminated in each environment state. In this example, options are defined as trying to reach each coin and the end state in the game. In other words, I, for the options that try to reach the coins, are the states in which those coins exist. On the other hand, I for the option trying to reach the end are the states which do not have any coin remaining on the map. The parameter β for each of these options is the state where the agent reaches the exact tile wherein the coin or the end is located. For each of these options, a separate DQN is defined that gets awarded for reaching its respective goal.

The agent is only permitted to use these options as its actions. After each option is terminated, a new option from the remaining pool of those available is selected. This arrangement was chosen to increase the speed at which the agent learns to finish the game.

4. The proposed approach

The proposed model in this paper uses a trained neural network to generate levels that have balanced parameters. In other words, if one inputs a vector with random values to the neural network, the output will be a balanced level. In this section, the general framework is first presented and then the detailed algorithms of the proposed model.

As for the first step, a Mario-like 2D platformer game has been designed using the Python programming language. The levels of the game contain the starting point, ending point, platforms, and coins. The rule of the game is that the player starts from the starting point. Then, they must obtain all of the coins. In the end, after receiving every coin, the player should go to the ending point to win and pass the level. If the player fails to finish the game in 300 cycles or falls down from a platform, they lose. As a reinforcement learning algorithm has been used to identify whether a level is solvable after a certain amount of repetitions

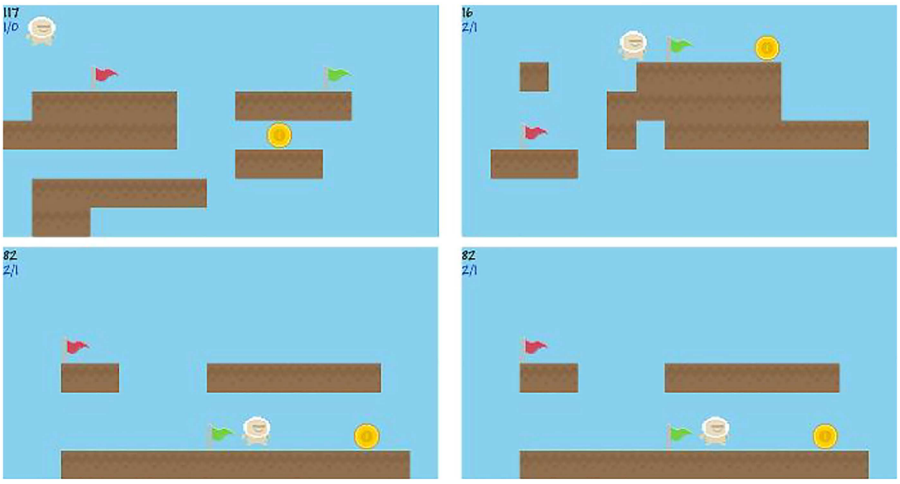

Figure 6. The designed 2D platformer game for the proposed approach.

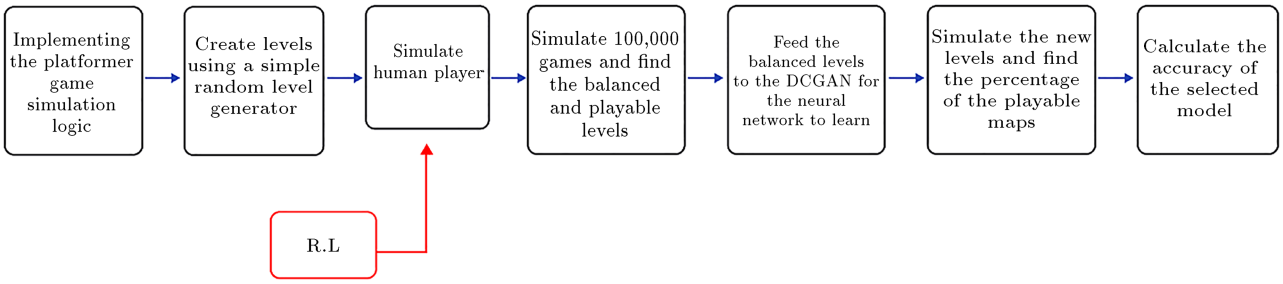

Figure 7. The general layout of the introduced framework.

or not, earning coins or reaching the endpoint has a positive score, while failure creates a negative score. For every positive score, the AI learns the pattern and will try to apply it in the next steps. In the next step, a series of simulations are performed. If the level is solved at least once in 300 attempts, this level is stored as a balanced solvable level. Otherwise, it will be stored as an un-solvable and unbalanced level. The 300 number is the selected threshold used in this paper. This threshold can be easily modified by different game designers based on their needs and requirements. These solvable levels are fed to the GAN network so that it can start generating similar levels with the same overall characteristics, and, hence, satisfying the aim for automatically creating balanced levels.

4.1. The general framework

The GAN requires a very large number of input data points. To achieve this, it is required that a game be designed whose levels could be used as the neural network inputs. Therefore, a two-dimensional 2D platformer game is designed for this purpose. The logic of the game is written using the Python programming language. In addition, Pygame is used to display the logic of the game if the user has enabled the graphical implementation of the game's settings. The simple 2D platformer game designed for this research is shown in Figure 6.

The general layout of the proposed approach is shown in Figure 7. As can be seen in the figure, the game's map is read from a bitmap image with a size of 15×8 pixels. Every pixel's RGB values represent the status of that cell in the game. At the beginning of the game, the level is generated based on the pixels of the map written using Algorithm 2, as shown in Figure 8. The player in the game can go *right*, *left* or *jump*. The jump can only be performed when the player is standing on a platform. The jump speed and the player speed can be changed in the game configuration file.

The logic of the game at any time is calculated according to the user's inputs, as well as the previous state of the game map. For example, if the user pushes constantly to the right, during that time cycle the player will move based on the player speed in the configuration file. Other aspects, such as game physics, collision with coins or endpoints, gravity calculations and anything that requires game logic is performed in this cycle, and then the state of the game will be transitioned to the next time cycle. In the rest of this section, the primary stages will be explained in detail.

4.2. Level generation

In order to obtain balanced levels, it is necessary to

```
Input:
 (1) C: number of coins
 (2) X: number of row slots
 (3) Y: number of column slots
Output:
 (1) An image in size of x and y with c coins and one starting flag and one
     ending flag
_____________________________________________________________

  PROCEDURE:
1:  | Compute randVar value from a random value between 4 and 8.
2:  | FOR (i = 0; i < randVar; i++)
3:  | |  Select random point in the map that is free.
4:  | |  Create random horizontal platforms of between 0 and 4 starting from
    | |  that point.
5:  | END-FOR
6:  |    Select a random point in the map exactly above a platform that is free
    |    and set it as the starting flag.
7:  |    Select a random point in the map exactly above a platform that is free
    |    and set it as the ending flag.
8:  | FOR (i = 0; i < C; i++)
9:  | |  Select a random point in the map one or two pixels above a platform
    | |  that is free.
10: | |  Place a coin on the selected point.
11: | END-FOR
  END-PROCEDURE
```

Figure 8. Algorithm 2: Level generation.

create levels in a random manner. Then, among the generated levels, select the ones that are balanced and use them as inputs for the neural network. Therefore, Algorithm 2 is used to randomly map the levels. These levels should have the necessary conditions for a playable game. This randomly generated level may be balanced, unbalanced, or even unsolvable. For this reason, reinforcement learning is applied to determine the level's solvability.

4.3. Level evaluation

Random levels should be considered in terms of solvability and balance. For simulating the way a real player progresses through a level, reinforcement learning has been used. That is, each randomly generated level is played 300 times. If among these 300 times, the level is solved even once, it is stored at a balanced level. Of course, this parameter is only selected as an intuitive baseline and can change depending on the actual requirements of the level designer.

Reinforcement learning ultimately tries to maximize the accumulated reward for the agent. If the rewards in the game are set up in such a way that reaching the coins or getting to the ending state while having no remaining coins, will result in a positive numerical reward for the agent, the reinforcement learning algorithm tries to maximize the reward and helps the agent finish the game. While the goal of this paper is not to create a player agent for the game environment, for the purpose of evaluating the created maps, such an agent can be used to see if a map can actually be solved or not within the stated number of repetitions. Therefore, a hierarchical reinforcement learning agent with an options framework and DQN as the baseline algorithm is set up in the game and given the task of finishing a given map. If the agent is able to finish the map in less than a number of episodes, the map is considered solvable. As there is no need to actually learn how to play the game, the first time an agent finishes the given map, the learning process is terminated.

The DQNs are defined as neural networks of four initial convolutional layers followed by two layers of multi-layer perceptron. The last layer has as many neurons as the actions the agent can use in the game, which are *left, right* and *jump*. Options are defined as explained in Section 3.5 and each of them has a DQN assigned to learn how to reach its specific goal.

By using this architecture, the chance of determining whether a map is solvable or not is increased, as previous successes in reaching intermediate goals such as getting coins are used to improve the performance of the agent in the next episode. Algorithm 3, shown in Figure 9, shows this architecture for evaluating a generated map. The architecture of the deep neural network used in the reinforcement learning algorithm of this study is also shown in Figure 10.

4.4. Learning level generation using GAN

After the balanced levels are saved, they should be fed to a suitable extension of GAN. For this research, the implemented model of deep convolutional GAN is considered. An Adam optimizer has been used with an input of 0.003. Also, training the network is performed using the binary cross-entropy method.

The assumed batch size of the network is 32 and

```
Input:
 (1) A generated map
 (2) K: number of training attempts
 (3) M: RL training episode
Output:
  (1) Whether the map is solvable or not
-------------------------------------------------------------------
  PROCEDURE:
1:  | Start the RL agent in the environment.
2:  | FOR (i = 0; i < M; i++)
3:  | | IF (the episode was completely successful)
4:  | | | Finish the training and add the map to the solvable levels
5:  | | END-IF
6:  | | IF (number of training episodes > K)
7:  | | | Stop the training and add the map to the unsolvable levels
8:  | | END-IF
9:  | | Perform RL training
10: | END-FOR
  END-PROCEDURE
```

Figure 9. Algorithm 3: Reinforcement learning algorithm for solvability.

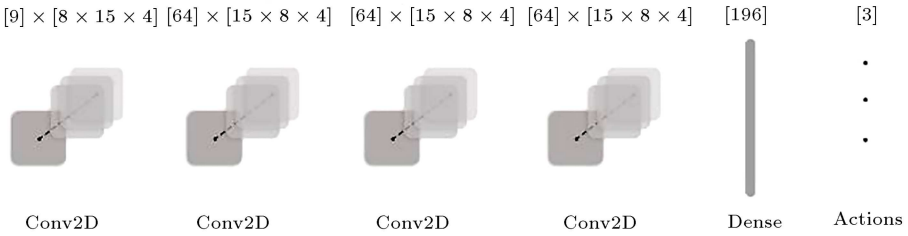

Figure 10. Reinforcement learning's neural network architecture.

for each dropout layer, it is considered to be 0.2. As discussed in Section 3, the GAN consists of two parts, namely, the generator and the discriminator. The GAN for this research has a different activation function on each layer, which is presented in detail in the following.

4.4.1. The generator

The generator is a neural network that receives noise data as input and delivers data to the output layer. The generator's neural network architecture used in this research is shown in Figure 11.

As shown in Figure 11, a 1×100 vector of the noise data is received randomly in the input layer of the generator. Then, in the next layer, a layer with $256 \times 15 \times 8$ neurons will be placed as the tanh activation function. In front of this layer, there exist three layers of $8 \times 15 \times 128$. The output layer is an $8 \times 15 \times 5$ convolution with a sigmoid activation function, which has the same image dimensions as those of the level in five different channels. In this research, each pixel color is considered as a channel. The result is a 5-channel neural network. In this way, when reading balanced maps to use as inputs, the input is stored as a 3D array. In this 3D array, the first dimension is the row, the second dimension is the column and the third dimension is the channel.

4.4.2. The discriminator

The discriminator is a neural network that receives the training data and determines whether the input data is real or fake. The discriminator's neural network architecture used in this work is shown in Figure 12. As shown in the figure, in the input layer of the discriminator, an 8×15 image that contains five

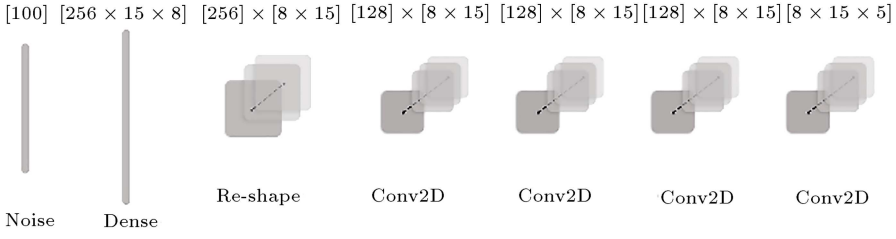

Figure 11. The generator's neural network architecture.

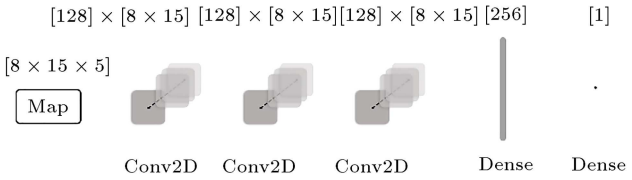

Figure 12. The discriminator's neural network architecture.

channels is received. This image, if provided as the training dataset, has a real data tag and if it is received from the output of the generator, it has a built-in label.

Then, in front of this layer, there are three layers with $128 \times 15 \times 8$ neurons with the activation function tanh. In the next layer, a layer with 256 neurons is placed as the tanh activation function. The output layer contains a neuron with a sigmoid activation function whose value is between 0 and 1. This number represents the extent to which this learning data belongs to the training data set or the output of the generator.

Algorithm 4, shown in Figure 13, represents the training algorithm for the GAN used in this research.

After the training process, as well as when allocation of the deep neural network weights is finished, a new level is created using Algorithm 5 shown in Figure 14. This level is a better candidate from the viewpoint of the discriminator.

Algorithms 4 and 5, respectively, perform the training process. In the end, a model that generates balanced levels is stored to be used in the level generation process. The summary of the steps taken in the proposed approach are as follows:

1. The generator constructs a random 100-digit vector between 0 and 1;
2. To perform a forward propagation, the output of each layer goes to the next layer. This will result in the final layer of the generator to create an image of the level;
3. The discriminator receives 32 images from the generator and 32 images from the training dataset as input;
4. The output of the discriminator's final layer, which is the prediction of the network from each data label, is computed in the form of forward propagation;

```
Input:
 (1) The level map in size 8*15
 (2) Iteration: number of training iterations
 (2) K: number of training steps
Output:
  (1) Trained weights for creating the level maps
_____________________________________________________
  PROCEDURE:
1:  | Start the RL agent in the environment.
2:  | FOR (i = 0; i < Iteration; i++)
3:  |   | FOR (j = 0; j < K; j++)
4:  |   |   | Sample mini-batch of m noise samples {z(1),...z(m)}.
5:  |   |   | Sample mini-batch of m examples {x(1),...,x(m)} from the dataset.
6:  |   |   | Update the discriminator by ascending its stochastic gradient.
7:  |   | END-FOR
8:  |   | Sample mini-batch of m noise samples {z(1),...,z(m)}.
9:  |   | Update the generator by descending its stochastic gradient.
10: | END-FOR
  END-PROCEDURE
```

Figure 13. Algorithm 4: Training the generative adversarial network.

```
Input:
 (1) Random numerical vectors of size 100 with values between 0 and 1
Output:
 (1) Level maps
_____________________________________________________
   PROCEDURE:
1:  | Give the input vectors to the generator and compute the output maps
2:  | Use the discriminator to get its opinion on the fakeness of the generated
    | maps
3:  | Sort the maps using the fakeness value
4:  | Select the maps that the discriminator considers the least fake.
   END-PROCEDURE
```

Figure 14. Algorithm 5: Generating the map.

5. The generator will keep its weights fixed and by calculating the error and using backward propagation, the weights of the discriminator are modified to better detect the real and fake data;
6. The generator produces a number of new noise data which is calculated by the forward propagation method;
7. Finally, the weights of the discriminator are kept constant. By calculating the error and using backward propagation, the weights of the generator are modified with the aim of producing data having the characteristics of the input data set.

After the completion of every 50 learning steps, the generator creates 25 levels. To generate any of these 25 steps, the generator first produces 20 levels and then the discriminator stores the level that is considered to be the best.

5. Evaluations and comparisons

In this section, the evaluation scenarios devised to demonstrate the applicability and accuracy of the proposed model are given. First, the simulation setup is discussed and then, the designed evaluation scenarios are given.

5.1. Simulation setup

The simulations for the 100,000 different levels were performed on a computer with a Core i7 7800 processor, an NVidia 1070 Ti graphic card, and 32GB of RAM for 40 days. From these 100,000 levels, 56,000 levels were stored as balanced and subsequently used as input data for the GAN. In another configuration, the same evaluations were performed using the cloud computing center at Iran University of Science and Technology [52]. 4 Core i7 7800 processors, 4 NVidia GTX 1080 (SLI), and 80 GB of RAM were provisioned. The evaluation time was reduced to 11 days.

Generally, training deep learning approaches using a huge dataset is a time-consuming process.

Most of the related work in this field has been faced with a similar challenge. For example, Frank et al., in their research, endeavoured to reduce the training time of their previous work which took more than 10 days to be completed [53]. In another related work, Berner et al. tried to use deep reinforcement learning approaches to learn the Dota 2 game and the learning time took more than 3 months to be finished [54].

Because the training process takes place on a large volume dataset, it is also intensive in terms of power consumption. The authors of this paper have applied a set of techniques to potentially control and reduce the amount of required power. These techniques are as follows:

1. In the proposed approach, the agent will top the learning process whenever it can solve the level in any of the attempts. Due to the provided definition of a balanced level in this work, the average learning time would be much faster compared to the case where more than one attempt was required to evaluate the balance characteristic for a level;
2. The agent will skip the level that is not solvable, which saves time for learning other remaining levels.

Finally, it is worth mentioning that the learning process is a pre-processing activity. After the learning part is completed, the identification of balanced levels will be performed significantly faster compared to a manual process. Such automation is the primary aim of the current research and also related research.

5.2. Evaluation scenarios

In the proposed approach, when the balanced levels are selected, they are then fed into the DCGAN architecture as the training input data. After about 170,000 epochs, the network begins generating levels that meet the minimum requirements. Also, 25 random levels in the form of images are generated and saved every 50 full training epochs. Figure 15 shows how the change in the output of the generator network during the training process is occurring.

As shown in the figure, the pixels of each step are set up completely random. After a small amount of training, the network learns that more pixels of the image should be white. It is also evident from how the platforms are placed in the image that the network is learning through time. In the early stages of the training process, the network cannot properly guess the number of cells for starting and ending points. But, as the training progresses, the network's advancement

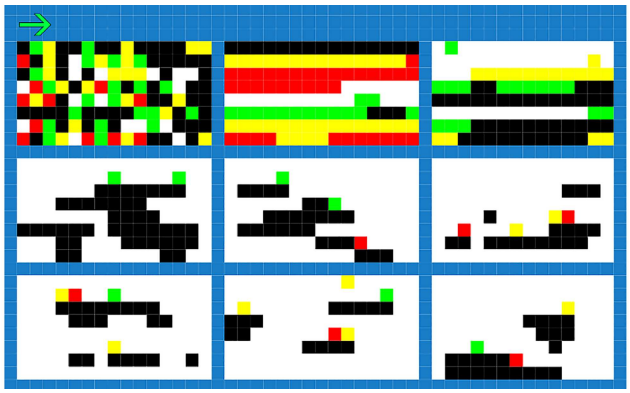

Figure 15. The progress being made in the generated level from the starting epoch to the end. The top left image shows the first epoch which is completely random. As the training progresses, better levels are generated both from a logical and balance points of view. In the final epoch shown as the bottom right image, a well-formed final level is generated.

is significant. As the training network grows further, the number of starting and ending cells reduces until the point where only one set of starting and ending points are generated in the later stages, and coins are also modified to the right size. In the figure, the yellow squares denote the coins, black squares are the platforms, green and red squares denote the starting and ending points, respectively, and white squares are the empty spaces. The network does not initially generate levels that have the exact requirements given in Algorithm 2, as shown in Figure 8. In other words, as the main approach of this work is to use GAN networks as an inherently unsupervised learning approach, such adherence will take place after the training process is completed, both for the generator and discriminator components. Hence, initially, there is a possibility that the generated maps from the GAN network have 3 or 4 coins, or that the number of platforms differ in count, as given in Algorithm 2. But, such differences do not fail the main purpose of this paper. Eventually, and after the training process is completed, levels with the stated conditions will be generated.

After completing the network's training process, stored models are used to assess the condition of the level. As mentioned, the randomly generated levels have a series of basic rules. The minimum requirements of a map in the game are:

1. It should have exactly one starting point;
2. It should have exactly one ending point;
3. There must be at least 1 and, at most, 3 coins on the playing field;
4. There must be at least 4 platforms;
5. The level must be solvable between 10 to 300 episodes of an RL assisted search. In other words, in order to keep the difficulty balance of the created level and prevent the level from being too easy or too difficult, the condition for assuming a level to be balanced is considered to be the number of attempts that the RL agent makes to solve the level. Hence, if the agent solves the created level in less than 10 tries, the level is considered to be too easy, and if the number of tries to solve the level gets above 300, the level is assumed to be too difficult to solve. Of course, these threshold numbers can be changed and modified by the level designer to get the best outcome.

For every 1000 epochs of training, 200 levels are created by the generator. Each of these levels is selected by generating 20 new levels using the generator and picking the best one using the discriminator. Each of these 200 levels is checked to see if they meet minimum requirements. The result of this assessment is shown in Figure 16. Based on this figure, it can be seen that the minimum requirements are met quite early in the training of the GAN and it reaches a 95 percent adaptation rate to level requirements after completing about 40000 training epochs. On the other hand, as seen in Figures 17–19, it takes more time before the average accuracy of the GAN model peaks. Also, meeting the minimum requirements is not the same as generating balanced levels. So, another evaluation step is necessary to assess this approach.

For evaluating the solvability of the produced levels, after every 10000 training epochs, the model is saved and the generated levels are checked for

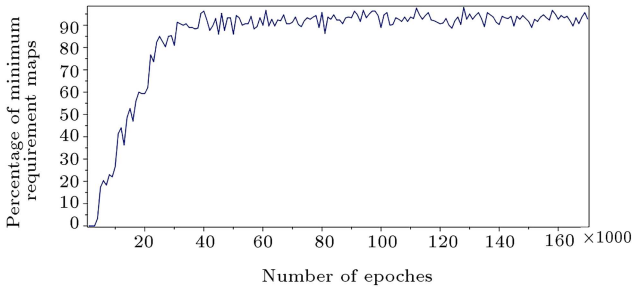

Figure 16. The percentage of generated maps that meet the minimum requirements per each 1000 epoch.

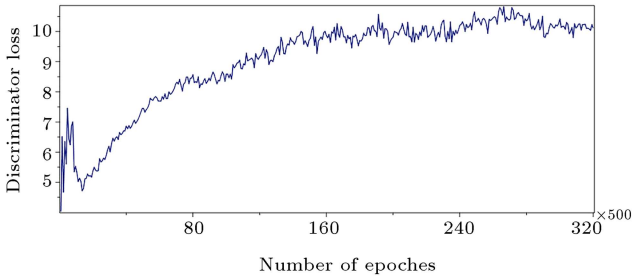

Figure 17. Discriminator loss in DCGAN training epochs.

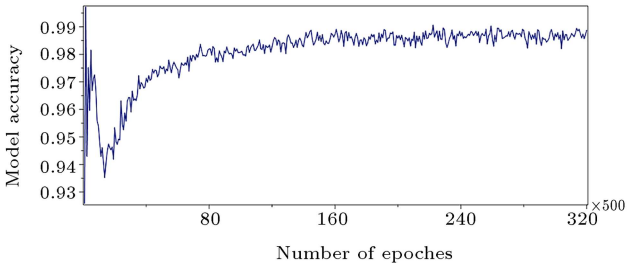

Figure 18. Model accuracy average in DCGAN training epochs.

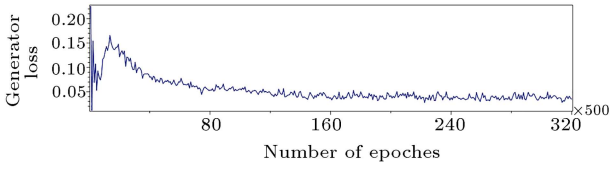

Figure 19. Generator loss average in DCGAN training epochs.

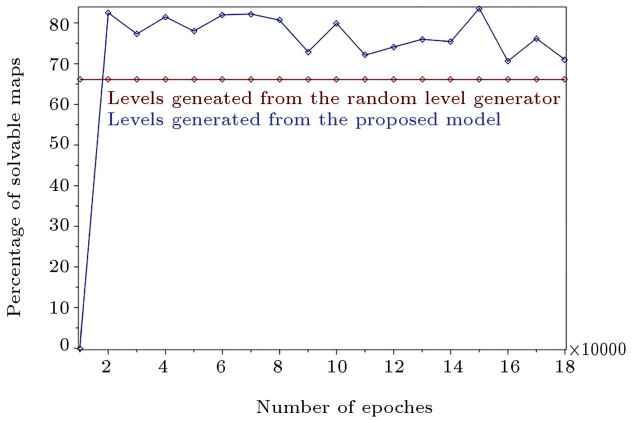

Figure 20. The comparison between the balanced levels generated from the proposed model and a random level generator.

solvability. Therefore, these models, with the generator and the discriminator network, are placed alongside the simulation file. Then, instead of getting the level from the code, the levels are obtained from the generator model. Getting the level from the trained model is performed by generating 20 levels. The discriminator selects the best level amongst those. Then, if the selected level has the necessary conditions to be used as a level, it will be used in the game itself. Otherwise, this process continues until the level that has all the requirements for being a level is selected.

To evaluate the performance of the GAN used in this paper, the model state in every 10000 training epochs is used to create potentially solvable levels. In this way, 17 models are selected from the stored models. Then, from each of these models, 2000 levels are generated and the solvability rate is calculated. In addition, and for the sake of having a baseline for comparison, the game has been executed 2000 times using a random level generated by a random level generator. Figure 20 shows the percentage of the balanced levels created using both of the level generators. As shown in Figure 20, the best rates of the solvable levels are in the model saved after the 150000 s training epoch, in which approximately 83.6 percent of the levels that the generator produces are solvable and balanced. It is also observed that the solvability rate of the levels generated using the best-learned model has improved by 17.5% compared to the random level generator.

6. Final discussions

One of the points mentioned among the advantages of the proposed approach is the non-existence of human operator involvement in the level generation process. In other words, a large set of random levels are fed to the GAN to generate the appropriate levels based on the level requirements specified by the level designer. This choice is made based on the fact that creating human-generated levels is both time-consuming and costly. It is important to note that there is no claim made that the proposed approach has better accuracy compared to the manually generated levels using a human operator and can replace this manual process. The primary advantage of the approach in an applied context is that it can provide the level designer with a huge number of balanced levels from which she/he can select based on her/his preferences or taste. Of course, by changing the required parameters in different application contexts, the approach can be viewed as a potentially general mechanism for creating levels in various game genres.

7. Conclusions

Creating balanced levels in video games is one of the most important challenges for game and level designers. The lack of attention to the balance of the game may cause a major failure and a huge profit loss for the company. Thus, in order to automatically create balance in the game, and consequently reducing human-error in the design process, in this paper, an approach is proposed based on Generative Adversarial Networks (GANs). The proposed approach focuses on generating balanced levels in 2D platformer games. To do so, a level generator is created that generates random levels. Then, using reinforcement learning algorithms, the solvability of the levels is investigated. The levels that are identified as solvable are stored as potential candidates and are then used as inputs of the GAN's neural network. After completion of the training process, the neural network saves its best model and, in the end, it produces a series of balanced levels. The generated levels and their appropriateness are then evaluated and compared to a baseline random level generator. It is shown that the proposed approach using GAN can generate a balanced level with 83.6 percent accuracy. This result implies that if a level designer maps the levels into an image file and applies a GAN-based approach, such as the one presented in this work, she/he can build a new balanced level with high precision.

The approach used in this paper has the following strengths compared to the other commonly-used approaches in the respective field:

1. Because the input is fed as images, it can be used in other game genres in which the levels or parameters can be stored in an image format. Therefore, this research is not specific to the 2D platformer game genre;
2. The selection of the features of the neural network is not performed by a human operator. Thus, the probability of the existence of human error is low;
3. After training the neural network, the process

of generating a new balanced level is performed efficiently and with high speed.

This research can be improved in the future to a great extent. The suggested extensions believed to be possible for the future of this research are as follows:

1. Changing and improving the parameters and layers of the generative adversarial network may result in better outcomes. Modifications of the parameters and the number of layers, or even using another activation function that is better suited to this problem, can be considered as a natural next step;
2. Experimenting with the applicability of the method in a game from another genre. The suggested genres that seem to be a potential fit are dungeon crawlers, beat'em up games and even first-person shooters;
3. Using the proposed framework to generate not only levels but also other types of content, such as personalized units, balanced weapons, business model design and even texture and 3D model generation, are some of the exciting areas that can be explored;
4. In this research, there has been a tendency to generate levels using GAN networks with the balance conditions mentioned by game designers. The aim has not been on generating levels with the conditions given in the base random level generation algorithm (i.e. Algorithm 2). Creating a better base random level generator and trying to force the GAN network to generate levels that have exactly the same rules and conditions of the base level generator can be an interesting line of research for further expansion;
5. The reinforcement learning agent in this article has been used in order to check solvability and difficulty of levels in the shortest possible time. Because of the approach of the reinforcement learning algorithms, the results seem to be logical but it does not guarantee that all the levels will engage players or be fun for them. So, the games may follow the definition of balance used in this paper but may not be necessarily engaging or even fun. Putting more focus on the engagement metrics of the levels is an interesting subject to be explored in further expansions;
6. The network used in this paper takes a long time to check and distinguish balanced and unbalanced levels. Changing this network or using an additional pre-processing stage (for example identifying impossible levels immediately and filtering them before giving the generated levels to the RL algorithm) can improve the overall speed of the approach, making it more practical for use in more complex scenarios.

References

1. Jaffe, A., Miller, A., Andersen, E., et al. "Evaluating competitive game balance with restricted play", *8th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Stanford, California, U.S., pp. 26–31 (2012).
2. Lopez, S.J. and Snyder, C.R., *Oxford Handbook of Positive Psychology*, Oxford Library of Psychology, Oxford University Press, UK (2009).
3. Adams, E., *Fundamentals of Game Design*, Pearson Education, New York, U.S. (2014).
4. Kavanagh, W.J., Miller, A., Norman, G., et al. "Balancing turn-based games with chained strategy generation", *IEEE Transactions on Games* (2019). DOI: 10.1109/TG.2019.2943227
5. Pérez, L.J.F., Calla, L.A.R., Valente, L., et al. "Dynamic game difficulty balancing in real time using evolutionary fuzzy cognitive maps", *14th Brazilian Symposium Conference on Computer Games and Digital Entertainment (SBGames)*, Rio de Janeiro, Brazil, pp. 24–32 (2015).
6. Silva, M.P., do Nascimento Silva, V., and Chaimowicz, L. "Dynamic difficulty adjustment through an adaptive AI", *14th Brazilian Symposium Conference on Computer Games and Digital Entertainment (SBGames)*, Rio de Janeiro, Brazil, pp. 173–182 (2015).
7. Yu, X., He, S., Gao, Y., et al. "Dynamic difficulty adjustment of game AI for video game Dead-End", *3rd International Conference on Information Sciences and Interaction Sciences*, Chengdu, China, pp. 583–587 (2010).
8. Bosc, G., Tan, P., Boulicaut, J.F., et al. "A pattern mining approach to study strategy balance in RTS games", *IEEE Transactions on Computational Intelligence and AI in Games*, **9**(2), pp. 123–132 (2017).
9. Makin, O. and Bangay, S. "Orthogonal analysis of StarCraft II for game balance", *Australasian Computer Science Week Multiconference*, Geelong, Australia, Article No. 30 (2017).
10. Schell, J. "The Art of Game Design: A book of lenses", 2nd Edn, AK Peters/CRC Press, Florida, U.S. (2014).
11. Karavolos, D., Liapis, A., and Yannakakis, G. "Learning the patterns of balance in a multi-player shooter game", *12th International Conference on the Foundations of Digital Games*, New York, U.S., Article No. 70 (2017).
12. Olesen, J.K., Yannakakis, G.N., and Hallam, J. "Real-time challenge balance in an RTS game using rtNEAT", *2008 IEEE Symposium on Computational Intelligence and Games*, Perth, Australia, pp. 87–94 (2008).
13. Morosan, M. and Poli, R. "Evolving a designer-balanced neural network for Ms PacMan", *9th Computer Science and Electronic Engineering (CEEC)*, Colchester, England, pp. 100–105 (2017).

14. Bangay, S. and Makin, O. "Generating an attribute space for analyzing balance in single unit RTS game combat", *2014 IEEE Conference on Computational Intelligence and Games*, Dortmund, Germany, pp. 1–8 (2014).

15. Uriarte, A. and Ontanón, S. "Psmage: Balanced map generation for starcraft", *2013 IEEE Conference on Computational Intelligence in Games (CIG)*, Niagara Falls, Canada, pp. 1–8 (2013).

16. Goodfellow, I., Pouget-Abadie, J., Mirza, M., et al. "Generative adversarial nets", *Neural Information Processing Systems*, Montreal, Canada, pp. 2672–2680 (2014).

17. Liu, G.C., Wang, J., Youn, G., and Kim, J. "Multi-scale multi-class conditional generative adversarial network for handwritten character generation", *The Journal of Supercomputing*, **73**(12), pp. 1–19 (2017).

18. Tran, L., Yin, X., and Liu, X. "Disentangled representation learning gan for pose-invariant face recognition", *Computer Vision and Pattern Recognition*, **3**(6), pp. 1415–1424 (2017).

19. Song, F.Y.Z.S. and Xiao, A.S.J. "Construction of a large-scale image dataset using deep learning with humans in the loop", arXiv preprint arXiv:1506.03365 (2015).

20. Mathieu, M., Couprie, C., and LeCun, Y. "Deep multi-scale video prediction beyond mean square error", arXiv preprint arXiv:1511.05440 (2016).

21. Baddar, W.J., Gu, G., Lee, S., et al. "Dynamics transfer GAN: generating video by transferring arbitrary temporal dynamics from a source video to a single target image", arXiv preprint arXiv:1712.03534 (2017).

22. Yang, L.C., Chou, S.Y., and Yang, Y.H. "MidiNet: A convolutional generative adversarial network for symbolic-domain music generation", *18th International Society for Music Information Retrieval Conference (ISMIR'2017)*, Suzhou, China, pp. 1–8 (2017).

23. Sutton, R.S. and Barto, A.G., *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, England (2018).

24. Zhang, C., Vinyals, O., Munos, R., et al. "A study on overfitting in deep reinforcement learning", arXiv preprint arXiv: 1804.06893 (2018).

25. Summerville, A.J., Snodgrass, S., Mateas, M., et al. "The vglc: The video game level corpus", arXiv preprint arXiv:1606.07487 (2016).

26. Volz, V., Schrum, J., Liu, J., et al. "Evolving Mario levels in the latent space of a deep convolutional generative adversarial network", arXiv preprint arXiv:1805.00728 (2018).

27. Bontrager, P., Roy, A., Togelius, J., et al. "DeepMasterPrint: Fingerprint spoofing via latent variable evolution", arXiv preprint arXiv:1705.07386 (2017).

28. Hansen, N., Müller, S.D., and Koumoutsakos, P. "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)", *Evolutionary Computation*, **11**(1), pp. 1–18 (2003).

29. Shaker, N., Togelius, J., and Nelson, M.J., *Procedural Content Generation in Games*, Springer International Publishing, Switzerland (2016).

30. Togelius, J., Yannakakis, G.N., Stanley, K.O., et al. "Search-based procedural content generation: A taxonomy and survey", *IEEE Transactions on Computational Intelligence and AI in Games*, **3**(3), pp. 172–186 (2011).

31. McCormack, J. "Interactive evolution of L-system grammars for computer graphics modelling", *Complex Systems: From Biology to Computation*, ISO Press (1993).

32. Mizuno, K. and Nishihara, S. "Constructive generation of very hard 3-colorability instances", *Discrete Applied Mathematics*, **156**(2), pp. 218–229 (2008).

33. Belhadj, F. "Terrain modeling: a constrained fractal model", *5th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction*, Grahamstown, South Africa, pp. 197–204 (2007).

34. Papageorgiou, E.I. and Salmeron, J.L. "A review of fuzzy cognitive maps research during the last decade", *IEEE Transactions on Fuzzy Systems*, **21**(1), pp. 66–79 (2013).

35. Kyriakarakos, G., Dounis, A.I., Arvanitis, K.G., et al. "Design of a fuzzy cognitive maps variable-load energy management system for autonomous PV-reverse osmosis desalination systems: A simulation survey", *Applied Energy*, **187**, pp. 575–584 (2017).

36. Back, T., *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*, Oxford University Press (1996).

37. Crepinšek, M., Liu, S.H., and Mernik, M. "Exploration and exploitation in evolutionary algorithms: A survey", *ACM Computing Surveys*, **45**(3), pp. 1–33 (2013).

38. Morosan, M. and Poli, R. "Automated game balancing in Ms. PacMan and StarCraft using evolutionary algorithms", *European Conference the Applications of Evolutionary Computation*, Amsterdam, The Netherlands, pp. 377–392 (2017).

39. Espejo, P.G., Ventura, S., and Herrera, F. "A survey on the application of genetic programming to classification", *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, **40**(2), pp. 121–144 (2010).

40. Gunturu, M., Shakarad, G.N., and Singh, S. "Fitness function to find game equilibria using genetic algorithms", *6th International Conference on Advances in Computing, Communications and Informatics (ICACCI'17)*, University in Manipal, India, pp. 1531–1534 (2017).

41. Xia, W. and Anand, B. "Game balancing with ecosystem mechanism", *International Conference on Data Mining and Advanced Computing (SAPIENCE)*, Ernakulam, India, pp. 317–324 (2016).

42. Hendrikx, M., Meijer, S., Van Der Velden J, et al. "Procedural content generation for games: A survey", *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, **9**(1), pp. 1–24 (2013).

43. Kennedy, J. "Particle swarm optimization", *Encyclopedia of Machine Learning*, Springer, U.S. (2011).

44. Giusti, R., Hullett, K., and Whitehead, J. "Weapon design patterns in shooter games", *First Workshop on Design Patterns in Games*, Carolina, U.S., Article No. 3 (2012).

45. Cachia, W., Liapis, A., and Yannakakis, G.N. "Multilevel evolution of shooter levels", *11th Artificial Intelligence and Interactive Digital Entertainment Conference*, California, U.S., pp. 115–121 (2015).

46. Giacomello, E., Lanzi, P.L., and Loiacono, D. "DOOM level generation using generative adversarial networks", arXiv preprint arXiv:1804.09154 (2018).

47. Filatov, A., Filatov, A., Krinkin, K., et al. "2D SLAM quality evaluation methods", arXiv preprint arXiv:1708.02354 (2017).

48. Ponti, M.A., Ribeiro, L.S., Nazare, T.S., et al. "Generative adversarial networks", *Presentation Content Inspired by Ian Goodfellow's Tutorial on NIPS* (2016).

49. Calderone, D. and Sastry, S.S. "Markov decision process routing games", *8th International Conference on Cyber-Physical Systems (ICCPS)*, Pittsburgh, U.S., pp. 273–280 (2017).

50. Van Hasselt, H., Guez, A., and Silver, D. "Deep reinforcement learning with double Q-learning", *30th Artificial Intelligence Conference*, Phoenix, Arizona, U.S., pp. 2094–2100 (2016).

51. Kulkarni, T.D., Narasimhan, K., Saeedi, A., et al. "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation", *Advances in Neural Information Processing Systems Conference*, Barcelona, Spain, pp. 3675–3683 (2016).

52. Cloud Compuing Center, Iran University of Science and Technology, Available: https://ccc.iust.ac.ir/

53. He, F.S., Liu, Y., Schwing, A.G., et al. "Learning to play in a day: Faster deep reinforcement learning by optimality tightening", arXiv preprint arXiv:1611.0160 (2016).

54. Berner, C., Brockman, G., Chan, B., et al. "Dota 2 with large scale deep reinforcement learning", arXiv preprint arXiv:1912.06680 (2019).

Biographies

Morteza Rajabi obtained his BS and MS degrees in Computer Engineering from Iran University of Science and Technology. His research interests include the utilization of machine learning methods in the domain of video game development.

Mehrdad Ashtiani received his BS, MS and PhD degrees in Software Engineering in 2009 and 2011, and 2015, respectively, from Iran University of Science and Technology, Tehran, Iran, where he is currently Assistant Professor in the School of Computer Engineering. His main research interests include trust modeling and the applications of uncertainty modeling to the domain of computer science.

Behrouz Minaei-Bidgoli is currently Associate Professor in the School of Computer Engineering at Iran University of Science and Technology, Tehran, Iran. He is head of the Data Mining Lab (DML) that performs research on various areas in artificial intelligence and data mining, including text mining, web information extraction, and natural language processing.

Omid Davoodi completed his BS and MS degrees in Computer Engineering at Iran University of Science and Technology, Tehran, Iran and is currently a PhD candidate at Carleton University, Ottawa, Canada. His research interests include explainable artificial intelligence, reinforcement learning and procedural content generation using machine learning.