Chapter 3

Deep Learning for PCG of 2D video game levels

Eleonora Chitti
Department of Computer Science
Università degli Studi di Milano, Italy
eleonora.chitti@unimi.it
ORCID: 0000-0003-4369-4615

DOI: 10.54103/milanoup.282.c635

3.1 Abstract

Content generation is one of the most expensive and time consuming tasks in video game development, and Procedural Content Generation (PCG) approaches have been proposed to reduce costs since 1980 in video games as Elite. In recent years, Generative Adversarial Network (GANs) have shown promising results in providing new possibilities for procedural generation of new content, including generation of 2D game levels. However, GAN research for PCG is still an open issue, since the new levels procedurally generated should respect constraints of the gameplay rules and they should be playable. In this paper we will review the state of the art and explore three interesting approaches in literature for procedural generation of game levels with GANs. First, we examine how the data of human-designed video game levels can be represented inside data-sets suitable for GANs; in particular, we focus on the representation of the game rules associated with each object present in the level. Second, we expose three different approaches of PCG with GANs, and for each approach we deepen the proposed adaptations on the GANs to generate video games levels, the data-set used, and the algorithm's performance. For each approach, we also analyze the proposed evaluation methods for the procedural generated levels, in particular taking into account the *traversability*, verifying that the areas in the

level are reachable, and the *playability*, verifying that the goal target of a game level is reachable without excessive difficulty.

3.2 Introduction

Video games are multimedia products made of different game elements, 3D or 2D assets, musics, sound effects, text, game story, game levels. These elements can be split into the two categories of cosmetic content and functional elements. The cosmetic content refers to assets and graphics from artistic point of view. The functional elements can be defines as the *game space* [1] or the game content that is directly related to game mechanics and rules, that are the core of a video game. According to [2] video games can be evaluated examining human actions, since the player experiences fun when they spend energy to reach the goal respecting the game mechanics and rules. To ensure playability, the rules, established by game designers, must be respected also during the development of a video game: level design heavily relies on human expertise and expensive playtesting of the levels created; for example in "Fig. 3.1" the ending target in a Super Mario Bros 2D level should be reachable and the obstacles should be positioned correctly, not too close nor too far to avoid the creation of a level too difficult or even impossible to play, that leaves the fun aside by replacing it with frustration. Therefore, one of the main costs of developing video game is content creation, that requires non-trivial human effort. To reduce costs and human labor, algorithmic approaches for automatic content creation following the game rules, or Procedural Content Generation (PCG), have been proposed from 1980, PCG can have different use cases, as autonomous generation, repair of non-playable levels, and data compression. In the Rogue dungeon-crawling video game (1980)[1], the developers exploited PCG to generate novel content (dungeon rooms and hallways) every time the game was started. In the Elite (1984)[2], a video game of space simulation, the developers exploited PCG for data compression, they stored the seeds for a pseudo-random number generator instead of the sequence of numbers to generate the universe represented in the game with the star systems. Nowadays PCG with seeds for pseudo-random generation is still used in commercial games as No Man's Sky[3].

In recent years deep learning allowed to produce different types of content as audio, images, or 3D objects, therefore, its application to video games is a natural consequence: new approaches exploiting *the Adversarial Networks* have been proposed also for PCG. For example, generative adversarial networks GANs have been applied to generate background terrain environments of games [3] or 2D game levels [4]. However GANs, that work well for generating content for other creative

[1] https://www.mobygames.com/game/rogue
[2] https://www.mobygames.com/game/elite
[3] https://www.nomanssky.com

Figure 3.1 *Example of Super Mario Bros level: Mario has to reach the pipe-goal of the level.*

purposes, are not always applicable to game generation. Procedural generation of levels, as well as PCG of sprites, should output content coherent with the game purposes and requires specific game-play skills: the game levels should be playable and the sprites should represent images of specific characters doing an action allowed by game rules (for example jumping or fighting).

We propose a focus on existing deep learning methods with GANs to generate new functional elements of 2D video games, and in particular, new game levels. We will analyze three different approaches for level generation that expose interesting results and that we have considered interesting for those wishing to approach PCG of video game levels with GANs. In the first section, we will analyze in brief what are the GANs and how this framework with two neural networks can generate new artificial samples starting from a real set of images. In the second section, we will analyze possible representations of the characteristics of a video game level that can be used to create new data-sets feasible as input for GANs. In the sections 4-6, we will deepen three different GANs implementations to generate game levels, analyzing promising results and limitations of each approach. Finally, in section 7, we draw the conclusions .

3.3 GANs in brief

Generative Adversarial Network (GAN) defines a machine learning framework that allows to generate new artificial data that are plausible, as new images or audio, from a real or hand-made training set. It comprises two neural networks, (i) the generator **G**, and (ii) the discriminator **D**, that compete in a zero sum game, where one agent's gain is another agent's loss.

- **G** generates new data as similar as possible to the training set X_{true}: it receives in input the data as N-dimensional uniform random variables (noise) as input

and it learns to map the input to distribution of interest.
- **D** distinguishes candidates produced by the generator X_{gen} from the true data distribution: it receives in input the true training set and the new data generated by **G** and, for each data, it outputs the probability that it is true or generated.

Through this technique new data are generated with the same statistics as the training set: **G** generates candidates while **D** evaluates them. The core idea of a GAN is based on the training of the generator through the discriminator, that can tell how much realistic are the data generated. This means that the **G** is not trained to minimize the distance to a specific data, as an image, but rather to fool the discriminator. Both networks are in fact updated dynamically: independent backpropagation procedures are applied so that **G** produces better samples, while **D** becomes more skilled at recognizing the fakes.

To summarize, the main goal of **G** is to learn how to successfully fool the discriminator, while the goal of **D** is to learn how to better recognize the true data from mocked ones [5].

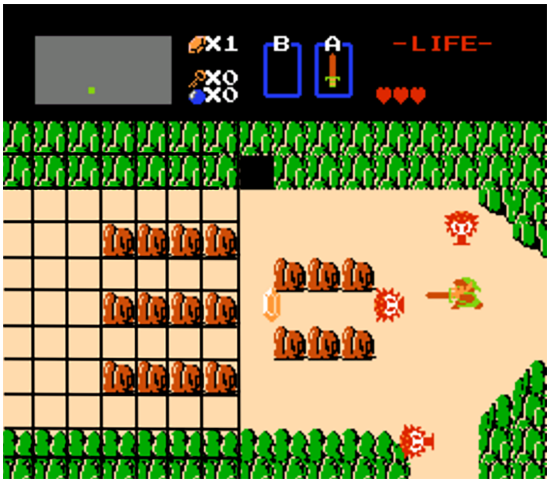

Figure 3.2 *Grid representation with tiles of Legend of Zelda. The map has walls and walkable areas, on the map are located tiles for the items (as rupees), obstacles (as stones) and enemies.*

3.4 2D Video Games data representation of levels for GANs.

Before exploring the deep learning approaches, it is important to make an introduction regarding 2D video games data representation. Most of 2D games can be represented with a *tileset*, literally a set of tiles, or sprite images, each one of the same size. These tiles can be placed on a grid to compose larger images, to be used for example as game maps. Each tile can contain a wall, a door, a walkable area or

the floor, a non-walkable area, an item or an enemy, as in “Fig. 3.2”. Tiles can be overlapped in particular cases, as images of enemies or items on top of the floor sprite.

3.4.1 VGDL representation

Starting from the tileset representation of 2D games, the *General Video Game Artificial Intelligence* (GVGAI) framework provides, through the *Video Game Description Language* (VGDL) [6], a generic solution to represent 2D video games with same physics and game actions and similar rules (most of them with top-down or isometric perspectives), e.g., “Fig. 3.3”. VGDL represents 2D games with a grid-level of description of its custom properties, constraints and physics: each sprite image inside a level in VGDL brings with it additional information as the directional speed, interaction with other sprites, its movement inside the level, declaration of the sprite as a game termination condition (for ending targets), scoring related to this sprite, etc.

The GVGAI framework allows to run 2D games defined with VGDL standard, and it provides agents based on various heuristics that can play the games. These agents have been exploited by different studies as [4,7] to evaluate generated levels.

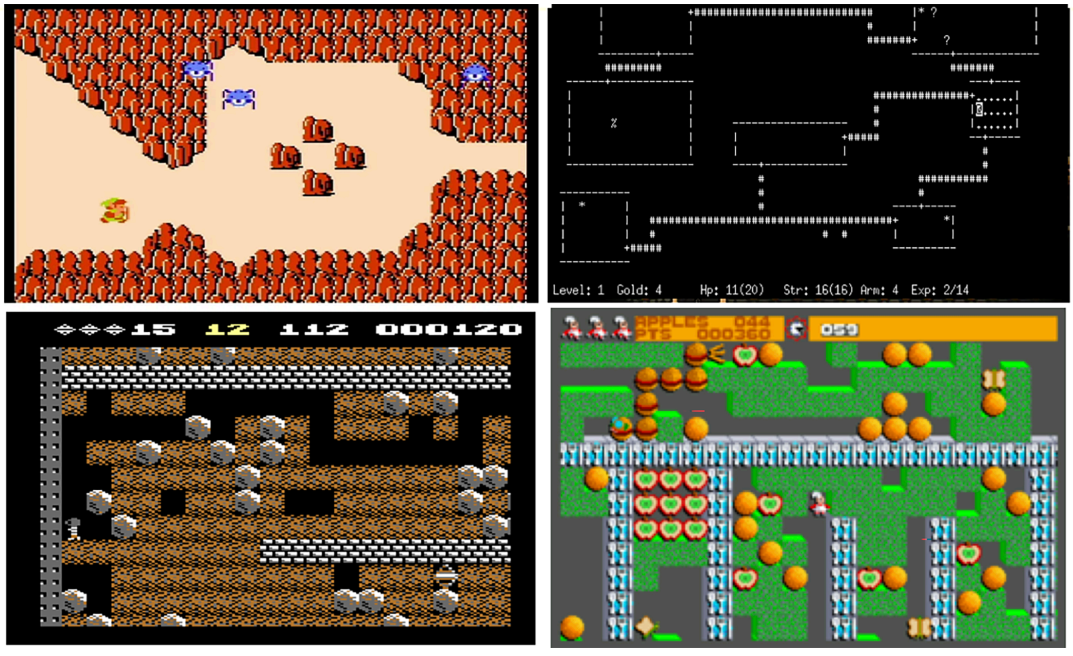

Figure 3.3 *Example set of four 2D games that can be represented with VGDL, from top left The Legend of Zelda, Rogue, Boulder-dash, Rockford.*

3.4.2 Maps representation

In the same year of the publication presenting the GVGAI, [8] proposed another approach defining a 2D game level with multiple overlapping images each one representing a feature of the level's map. This approach was explored for generating maps of first person games, in which the graphical perspective is rendered from the viewpoint of the player's character. The authors converted the walkable area of each level in top-down 2D maps, that can be used as a dataset.

The authors created 5 different images for each level's map:

- *Floor map*, representing floors.
- *Wall map*, representing walls.
- *Height map*, representing the height of floor of each room inside the level.
- *Things map*, representing all the objects included in the level that are not walls, floors or doors.
- *Trigger map*, representing all the triggers with different encoding, representing the type of trigger (door, teleport, lift or switch)

In addition they proposed a 6th image, called *Room map*, representing the map with a room segmentation approach similar to the one used to reconstruct maps of real life environments by robotic entities. These maps can be analyzed with the Simultaneous localization and mapping - SLAM algorithms, that can be used to evaluate the quality of the generated map.

3.5 DOOM levels generation with GANs

Authors in [8] propose and compare two different approaches to generate DOOM levels. In this study the authors applied GANs to train a model with existing DOOM levels and then they used the model to procedurally generate new levels.

3.5.1 Data-set

DOOM is a first person shooter video game developed in 1993[4], and its levels are widely represented in different textual data-sets as the Video Games Level Corpus by [9] and the *Idgames archive* [5]. [8] performed a preliminary conversion of data from text to images, they extracted topological features and converted data representing each level into 6 images, *(i) Floor map, (ii) Wall map, (iii) Height map, (iv) Things map, (v) Triggers map and (vi) Room map*, as described in the section 3.4.

The authors trained two different GANs with 1000 human-designed levels: the first GAN was trained with maps images, while the second one was trained using

[4] `https://it.wikipedia.org/wiki/Doom`

[5] `https://doomwiki.org/wiki/Idgames_archive`

both map images and a vector of features that were also extracted during the preliminary conversion of the data-sets. The vector of features include:

- *Metadata:* includes level name and number of downloads of the level
- *WAD*: the WAD is the name extension of files in the Idarchive data-set, these files include size of the level, number of lines and vertices, the same data are identified here with WAD term.
- *Image:* includes equivalent diameter of the level, fraction of area that can be traversed, perimeter of the level, vertical and horizontal size of the level
- *Graph:* graph representation extracted with the Room map image analyzing indoor environments with SLAM approach.

3.5.2 Deep Level Generation

The authors trained two different GANs I and II, one *unconditional* and one *conditional*. The authors employed the Wasserstein GAN with Gradient Penalty (WGAN-GP) developed by [10], they adapted the framework replacing the *tanh* activation function on the output layer, with a *sigmoid* function more suitable for grayscale images with few levels of gray.

The first GAN (I) is *unconditional*: for each level the generator G receives in input X_{true} **six images and a noise vector Z** sampled from a Gaussian distribution, while the second GAN (II) is *conditional*: the generator G gets in input the **the six images, a vector of features Y of the level and the noise vector Z**.

As a result, both the GANs generates six images that represent a generated level and can be used to produce a fully playable level. The G network is trained to generate an output X_{gen} that is as similar as possible to the original inputs X_{true}. In contrast, the D network is trained to distinguish between human-designed and generated levels, it receives in input either X_{true} or X_{gen} and it outputs the probability that X_{true} or X_{gen} are images of a generated level.

The authors trained the discriminator and generator networks following the approach proposed in [10], using Adam optimizer, they optimized the discriminator network five times for each update of the generator network. In each training iteration, they also applied a 90 degree clockwise rotation to the input images, so that the networks were trained using all the four possible level orientations. This transformation allows to exploit the rotation information in the representation of a DOOM level, since level's playability is not affected by its orientation in the space.

3.5.3 Generated Levels Evaluation

To evaluate the quality of samples generated by a GANs other studies proposed an approach based on human annotations or based on the score provided by an image classifier [11]. Instead in this paper [8], the authors designed a set of metrics inspired by the ones used to evaluate maps generated by the real world navigation

of robotic entities, that are very similar to the six maps that describe a DOOM level. The metrics proposed by [8] are:

- dE: It is the average absolute difference of entropy of the pixel distribution between human-designed and generated images. The authors selected the difference of entropy between true samples and generated samples to detect if the quantity of information encoded by the generated levels differs from the true samples.
- SSIM: average Structural Similarity index between the images of human-designed and generated levels, with values from 0 to 1, where value 1 represents a comparison of two images of the same map.
- Encoding Error - EE: measure of the average errors over the pixel values of the level images generated by the network (for each pixel there are few of meaningful color values, for example in the FloorMap each pixel should have either value black or white)
- CE: the average Corner Error between the images of human-designed levels and generated levels, that is a measure of how large is the difference between the average number of corners in the two sets of levels. This metric provides an estimated of the how close are human-designed and generated levels in terms of structural complexity. The author computed the CE's input value of number of corners through the Harris detector [12] for counting the corners contained in FloorMap and WallMap images.

The authors trained two GANs defined before as (I) *unconditional* and (II) *conditional*. In both networks I and II, the **discriminator loss** steadily decreases with training iterations and the loss achieved in the training set is close to the one achieved in the validation set, suggesting generalization capabilities.

For **dE of the FloorMap, WallMap and ThingsMap**, both networks I and II are able to generate images with dE values, not only close to 0 at the end of the training, but they are also able to generate images with entropy values very similar to the originals. For **dE of the HeightMap**, the generated images are more noisy, and none of the network is capable to reach values close to 0, although the II *conditional* network is capable to reach smaller dE that the I *unconditional*. For **SSIM and EE values** both GANs perform well generating images of good quality. In particular, the II obtains **SSIM** values of quality of images slightly better than I. For **CE values**, the levels generated by I do not improve over time, instead of CE values of levels generated by II slightly increase over time, suggesting that the levels generated by the *conditional* network are more complex and more similar to human-designed levels. The similarity can be examined also with visual comparison, since levels by the *conditional* network show a richer structure.

The authors proposed a novel approach to generate maps for the DOOM video games, the maps can be traversed since they have been examined using an evaluation approach similar to the one for SLAM algorithms. However, playability has

not been tested, and maybe the level are too easy or too difficult to play. Therefore, this paper has been an interesting starting point for future studies, as [7] that proposes an evaluation of the playability of generated levels. The approach in [7] will be analyzed below in this paper.

3.6 Bootstrapping Conditional GAN for level generation of The Legend of Zelda

[7] propose an approach with an adaptation of GAN called CESAGAN or Conditional Embedding Self-Attention GAN, to incorporate an embedding feature vector input to condition the training. The aim of the authors is to allow the network to model non-local dependencies between game objects that are fundamental for the playability of the level, for example a key that the player should pick up to pass a locked door. The authors then evaluate the playability of generated levels. In addition, not all the video game offer large data-sets as DOOM, so the authors propose a bootstrapping mechanism to overcome the lack of data, in this approach the new generated levels, that are marked as playable, are added to the training set. The authors observed that their approach does not only allow to generate a larger number of levels that are playable but also generates less duplicate levels compared to a standard GAN. The authors tested the framework developed generating new levels of The Legend of Zelda.

The Legend of Zelda (Nintendo, 1986) is an adventure video game in which the player needs to explore different dungeons, each dungeon is a level. The main goal of the player is to explore the dungeon, collect as many treasures and money as possible, find a key and reach the exit door of the dungeon without getting killed by the moving enemies. The player can kill the enemies using their sword to collect extra money.

3.6.1 Data-set

The GVGAI [6] is a framework built to run 2D arcade-like games written in VGDL language, described in section 3.4, and it offers a definition in VGDL of game rules pertaining each sprite present in The Legend of Zelda video game. The authors extended the VGDL definition of each sprite with the Identity information to be an additional input data for the GAN, as shown in Table. 3.1. The authors manually generated the data-set with 45 human-designed levels.

3.6.2 Deep Level Generation

According to the authors GAN-based models for level generation are built using convolutional layers, but it is not enough for level generation, since it is a local operation whose correlation depends on the spatial size of the kernel, and in an

Table 3.1 *VGDL encoding (Symbol) and encoding for GAN (Identity) of The Legend of Zelda elements*

Object	Symbol	Identity
Wall	w	0
Empty	.	1
Key	+	2
Exit	g	3
Enemy 1	1	4
Enemy 2	2	5
Enemy 3	3	6
Player	A	7

operation for level generation, it is not likely for an output on the top-left position to have any correlation to the output at bottom-right. So, to increase the search space, a deep convolution network with many layers would be required. In fact video games levels have correlations of items located far from each other, as the key and the door, and these correlations are fundamental to make the game playable.

Therefore the authors propose a new approach called CESAGAN, combining the methods of the self-attention GAN (SAGAN) proposed by [13] and the GAN adapted with the input vector proposed by [8].

The SAGAN allows to keep balance between efficiency and to capture long-range dependencies, and it is based on three different vectors: query (f), key (g) and value (h) which are three different mappings (output of a single perceptron neural network) of the input data (e.g. an image). The query and key undergo a matrix multiplication then pass through a softmax, which converts the resulting vector into a probability distribution attention map. This attention map determines the weight of each of the tiles and keep it in memory. Finally, the attention map is multiplied by the value to determine the relationship at a position in a sequence by attending to all positions within the same sequence.

The authors of [7] concatenate the feature embedding mapping and the self-attention feature map to combine SAGAN with the conditional vector representation u. This network is applied to both the generator (G) and discriminator (D). In addition the authors propose also a Bootstrapping mechanism to improve the CE-SAGAN (Fig. 3.4): after each epoch, a new set of levels is generated, and a playability analysis is carried out to identify the playable levels. The levels identified as playable are then added to the training set. Playability is evaluated with the following heuristics:

- There is only one player's avatar
- There is only one key
- There is only one door
- Enemies are less than 60% of the empty space

- The player can reach the key using an A* algorithm
- The player can reach the door using an A* algorithm
- The level has a border to prevent all the characters to go outside the level.

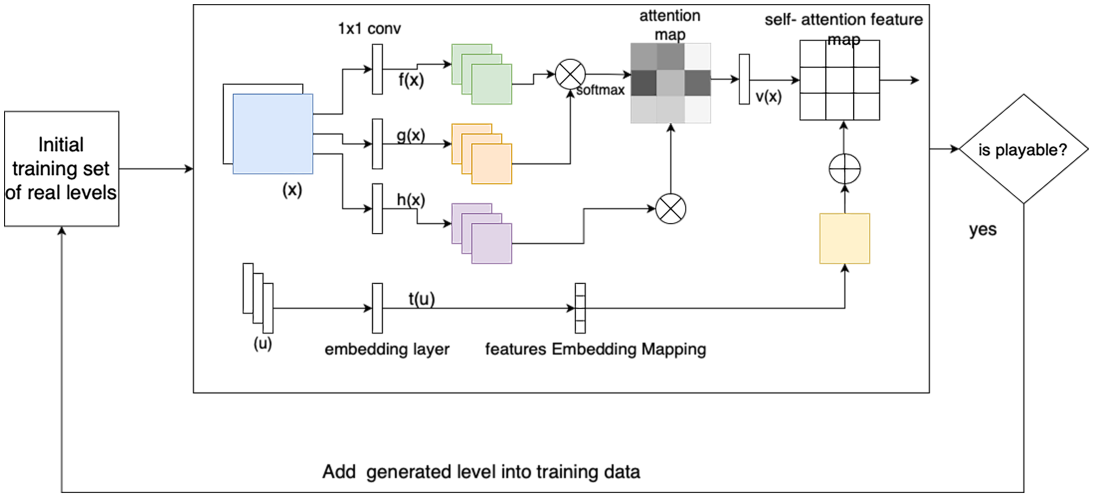

Figure 3.4 *Representation of the CESAGAN with bootstrapping mechanism.*

The authors suggest for future studies to substitute the proposed heuristics with an agent from the GVGAI framework to check for playability.

3.6.3 Generated Levels Evaluation

The authors tested their approach analyzing playability of generated levels and searching for duplicates of the generated samples. First, the authors tested the CESAGAN without the bootstrapping with the data-set with 45 levels and then compared the results with the state-of-the-art techniques. The CESAGAN performed better in generating playable levels, 58% against the 19.4% of the baseline, and it performed slightly better in not generating duplicates levels, 37.6% of duplicates against 39.4% of the baseline.

Second, the authors tested the CESAGAN with bootsrtapping with a data-set of only 5 human-designed levels. The authors motivated the decision of using a so small data-set by reporting that in to the state of the art data-sets for some type of 2D video games may contain few samples, and they want to propose an approach feasible for video games with few levels available. In any case, this second CESAGAN performed well with 47% against the 24.6% of the baseline of playable levels generated, and with 60.3% of duplicates against 98% of the baseline of duplicated levels generated.

3.7 Level generation for multiple types of 2D video games with common latent space with VGDL

[4] trained a GAN to generate game levels for four 2D video games with similar gameplay rules.

1. Boulder-dash
   Boulder Dash takes place in a series of caves, each one is a level designed as rectangular grid of blocks. The player can move freely to collect as many diamonds as possible while avoiding dangers, such as falling rocks.
2. Link
   Link is a strategy game with Roguelike elements (turn taking). The player controls a group of soldiers trying to survive on the hostile, arctic planet, crafting items, and collecting basic materials and advanced tools.
3. The Legend of Zelda
   The Legend of Zelda is an adventure game in which the player will explore different dungeons, each dungeon is full of monsters. The player is free to move and the main goal of the player is to explore the dungeon, avoiding or killing monsters, collecting treasures and money, while searching for the key to exit through a locked door.
4. Roguelike game
   In Rogue, players control a character as they explore several levels of a dungeon seeking the Amulet of Yendor located in the dungeon's lowest level. The player-character must fend off an array of monsters that roam the dungeons. Along the way, players can collect treasures that can help them offensively or defensively, such as weapons, armor, potions, scrolls, and other magical items. Rogue is turn-based, taking place on a square grid represented in ASCII or other fixed character set, allowing players to have time to determine the best move to survive. Rogue has been defined a masterpiece and it inspired the development of many games with similar mechanics, defined as Roguelike.

These four games have completely different gameplays, but they have in common the actions that the player performs to reach the goal, that are defined as the **sequences of actions** needed to complete successfully a level. For each game the same action leads to a different visual consequence but in all the games it permits to advance in the level: pick a bomb and break a wall is different from pick a key and open a locked door, but the same pick action leads to advance in the level. In fact the games are not equal considering the interaction of player with dynamic elements (as enemies, treasures...) that are different for each game.

Only *Boulder-dash* has falling boulders which obey gravity, while in *Roguelike* and *Zelda* have inside the map solid walls with locked doors or breakable walls that

require the player to first pick up a key or a bomb to pass. The ability of enemies, (soldiers, magicians, ...) are also unique for each game, and these differences result in a variety of sprite patterns.

3.7.1 Data-set

The VGDL language (section 3.4) defines for each sprite the particular properties, as scoring and movements allowed for that sprite, that can be useful for example to define enemies properties for each game. These four games have been already encoded in VGDL language and VGDL definitions is available for these games within the GVGAI framework [6].

The authors generated a new training data-set converting the **sequence of actions** along time in game levels, exploiting VGDL information on sprites. To generate the level from actions, the objective is to place obstacles that complement the actions at the right time and in the right location, such that the player action in the game timeline is necessary to move forward in a game.

A training sample is created by the authors through filling an empty grid: the starting point is on the top left quadrant and the goal point is chosen on the bottom right. A sequence of actions is selected, and each action will take the players avatar from one grid point to the next until the end. The action, as jump or break a wall, is carried out through the grid as sprite, as a ditch or a wall, that is placed in the way to match the corresponding action. The action sequences are varied by changing the order of actions or permuting the combinations of actions randomly. Multiple combinations of actions that take the players avatar from the start to the goal state are considered. The same action sequence is used in all the four games considered, but the specific game rules requires to place different types of obstacles to match the action. For example, in Boulder-dash one has to avoid falling boulders and in Roguelike one needs to first pick up a key before passing a locked gate. The approach is generic and can be used to generate levels for multiple games starting from a common action sequence and path through the grid. The training level generation algorithm is available at [4].

Training sets are generated explicitly with the same action sequence for all four game levels resulting in similar gameplay for all four levels. **Each level is represented as a multi-channel binary matrix**, with each channel representing one type of sprite in the game and each grid point being a binary representation of the presence of the corresponding sprite at that grid point.

3.7.2 Deep Level Generation

The authors propose a new approach of GAN to generate four different outputs with the same random seed, and each output corresponds to a level of the four games. This GAN has one Generator G and four Discriminators D, one for each

video game. Each discriminator is, in fact, tied to a single game and distinguishes between generated samples and original.

According to the authors this GAN will capture information that traditional GANs will not: the traditional GAN would not learn any common patterns as the only common element, the random input from the latent space, cannot be trained. In this new version of GAN the latent space and unbranched layers capture the commonality across all the four games while the branched D layers capture the differences, since D networks are trained independently one from the another.

Binary cross entropy loss from the discriminators is added to conditional loss from the generator. The generator loss is the sum of the binary cross entropy between the training sample and the generated image along with conditional loss if the number of sprites does not match the training level. The generators use batch normalization between convolution layers and LeakyReLU activation along with a final sigmoid activation to generate game level output. Each of the discriminators use a dropout of 30% to reduce overfitting. The generator generates a grid of size 16x16 for each game, from an initial input of 128 normally distributed random numbers. The training sample and generated game levels are represented as a tensor with nine channels, one each for each type of sprite (avatar, exit, floor, gold/health, key, lock, monster, wall and weapon). Unused channels are set to zero.

3.7.3 Generated Levels Evaluation

To evaluate the quality of samples generated by a GANs the authors exploited the agents from the GVGAI framework, each agent will solve the game ensuring that it is playable. The authors not evaluated only playability (or solvability), but also path similarity and novelty of the levels, defined as follows:

- **Solvability**: a level is considered as playable if a GVGAI agent can solve it at least once in 5 attempt.
- **Path Similarity**: this value defines if the GAN model has captured the similarity between the games, and it is calculated between game levels of distinct games generated together. If the GAN captures the gameplay similarity between distinct games, the distribution of path similarity distance should be the same between the training set and generated sets. A shortest path is calculated from the starting point to the goal. the shortest path do not ensure playability, since authors applied a Dijkstra shortest path algorithm, that takes into account stops on the grid cells with sprites of items that are necessary to step further in the level (as a key), that are considered as mandatory nodes to be traversed in the graph. The path similarity distance is the Manhattan distance between the grid locations in the path, and it is calculated between the shortest paths game levels generated together.

- **Novelty**: it is a measure of similarity within a game. A level is novel with respect to another if the path taken by the avatar, represented by the sequence of actions is different. To evaluate novelty, the authors used the Levenshtein distance (a string metric for measuring the difference between two sequences) between two action sequences. If the distance is large, the two levels are different: for example, a solution action sequence for a Zelda level is *(right, right, pick key, up, right, right)* and another level has actions *(right, up, pick key, up, right)* the distance between the levels is two: the number of changes inside one sequence to obtain the other.

The authors evaluate solvability of the generated levels by taking a set of 50 GAN's generated levels for each game and running an automated agent provided by the GVGAI framework. The Boulder-dash has the higher solvability of 70% over all the other games, probably because this game has the lowest number of obstacles (this game has not locked door or similar) so it does not have any dependency between actions. Probably because of the same reason the Link and the Roguelike are around 55%, and the Legend of Zelda has the lowest solvability with a score of 40%, since Zelda is the game with the highest number of obstacles, including locked doors, walls to broke, switches to activate and ditches to cross.

To evaluate the Path Similarity of gameplay across games for generated game-level sets, the authors plotted the average similarity distance between the ideal path for avatar to reach the end state from the start state, picking up the necessary items and ignoring dynamic items as monsters. The authors evaluated the distribution with the Wasserstein distance, reporting that the generated sets of four levels have path similarity distribution closer to the training set(Wasserstein distance value: 161) and further less that the baseline (Wasserstein distance value: 283), stating that the GAN has captured aspects of gameplay similarity across the four games.

Looking at the novelty values the authors reported that the generator has captured most of the complexity of the training set in its model. In fact, the variety of levels in the training set is captured by the GAN as the generated levels have a similar distribution of values for the Levenshtein distance.

The authors propose an innovative approach, that can be considered an interesting starting point to generate new levels procedurally. However, future studies should overcome the difficulty of achieving a better value of solvability of generated levels of games with more complex rules and obstacles as the Legend of Zelda.

3.8 Conclusion

We have shown three representative examples of Video Games Levels generation which use GANs to create procedurally new game levels that can be playable. In particular we described the problems of defining the levels with a representation feasible as input for GAN, and the two major approaches are: the representation of

a level wih 6 images each one representing a feature of the level's map, (as walls, triggers or collectable items) and the representation of a level with the VGDL language [6] that allows to encode for each image, that represents an element of the level (as a treasure), its features and its role within the game rules. We then analyzed use cases of the Video Games Levels generation, in particular in the area of 2D arcade-like video games. [8] compared two different approaches to generate DOOM levels, in the first approach the authors trained a GAN with a data-set of DOOM levels represented with the 6 maps images, and in the second one the authors adapted the GAN to take in input the same data-set and in addition a vector of level's features, however ths approach do not take into account the evaluation of playability of levels. [7] proposed an approach to procedurally generate levels of 2D video games for which large data-sets are not available, extending [8] approach with a [13] and a bootstrapping mechanism in which generated levels, that are marked as playable by an heuristic, are added to the training set. The results are interesting and future studies are needed to ensure the feasibility of using this framework with little data-sets. Authors in [4] propose the most recent approach, with the aim of developing levels of four different 2D games with a similar gameplay with only one Deep Learning Framework, that outputs four different generated levels (on for each game). Within this framework a Generator G generates all the 4 samples, that are then analyzed by 4 Discriminators D, each one taking in input only one sample relative to a type of game. Results are promising, however the performance of the network vary consistently between the four games, and the algorithms show lower performances in the generation of levels of the games which include more items in the gameplay, than the others games with less items, even if all the four games have similar rules.

Bibliography

[1] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup, "Procedural content generation for games: A survey," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 9, no. 1, feb 2013.

[2] E. J. Aarseth, "Cybertext: Perspectives on ergodic literature," 1997.

[3] E. Panagiotou and E. Charou, "Procedural 3D terrain generation using generative adversarial networks," 2020. [Online]. Available: https://arxiv.org/abs/2010.06411

[4] V. Kumaran, B. Mott, and J. Lester, "Generating game levels for multiple distinct games with a common latent space," *Proc. of the AAAI Conf. on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, no. 1, pp. 109–115, Oct. 2020.

[5] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014. [Online]. Available: https://arxiv.org/abs/1406.2661

[6] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, "General video game ai: a multi-track framework for evaluating agents, games and content generation algorithms," 2018. [Online]. Available: https://arxiv.org/abs/1802.10363

[7] R. R. Torrado, A. Khalifa, M. C. Green, N. Justesen, S. Risi, and J. Togelius, "Bootstrapping conditional gans for video game level generation," 2019. [Online]. Available: https://arxiv.org/abs/1910.01603

[8] E. Giacomello, P. L. Lanzi, and D. Loiacono, "DOOM level generation using generative adversarial networks," 2018. [Online]. Available: https://arxiv.org/abs/1804.09154

[9] A. J. Summerville, S. Snodgrass, M. Mateas, and S. Ontanon, "The vglc: The video game level corpus," 2016. [Online]. Available: https://arxiv.org/abs/1606.07487

[10] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," 2017. [Online]. Available: https://arxiv.org/abs/1704.00028

[11] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," 2016. [Online]. Available: https://arxiv.org/abs/1606.03498

[12] C. G. Harris and M. J. Stephens, "A combined corner and edge detector," in *Alvey Vision Conf.*, 1988.

[13] J. Cheng, L. Dong, and M. Lapata, "Long short-term memory-networks for machine reading," 2016. [Online]. Available: https://arxiv.org/abs/1601.06733