Procedural Generation of Game Levels and Maps: A Review

Tianhan Gao
Software College
Northeastern University
Shenyang, China
gaoth@mail.neu.edu.cn

Jin Zhang
Software College
Northeastern University
Shenyang, China
2071344@stu.neu.edu.cn

Qingwei Mi
Software College
Northeastern University
Shenyang, China
2110491@stu.neu.edu.cn

***Abstract*—So far, physical labor has ensured that the quality and quantity of game content match the needs of the game community. However, due to the exponential growth of gaming population and production costs in the past decade, it is facing new scalability challenges. Procedural Content Generation (PCG) can meet these challenges by generating game content in a fully autonomous or hybrid-led manner. Game level and map generation is a sub-field of PCG. In this paper, we summarize the ideas and main processes of the procedural generation methods of game levels and maps for racing games, platform games, and open world games, analyze the current development. Finally, we conclude the cognition and prospects of the procedural generation of game levels and maps.**

Keywords—procedural content generation; game levels and maps; game artificial intelligence

I. INTRODUCTION

Procedural content generation (PCG) [1] is a hot sub-field of game AI, which refers to completely autonomous or limited human-controlled methods for generating game content. PCG can help designers create content and enhance the creativity of individual creators, which eliminates the needs of human work. PCG can also create a new type of game that does not end. Combined with player modeling, PCG can create adaptive player games. Through code verification, PCG can help researchers understand the designing process and creativity. Game content is the key to ensuring the player experience, covering levels, maps, rules, maps, plots, items, tasks, music, weapons, vehicles, characters and other detailed branches. Lisapis et al. [2] have identified six creative areas in the game, including levels and maps, auditory effects, visual effects, rules, narratives, and the game itself. This paper will focus on the procedural generation of game levels and maps.

Game levels and maps generation is the most popular area of PCG so far [3] because in the game, levels and maps are equally essential elements as the rules of the game and are the main ways to drive players to interact. Different levels and maps designed under a fixed game mechanism will change the gameplay and player experience. PCG can generate two-dimensional images of Super Mario Bros. simple platform game levels; it can also generate the constrained two-dimensional space in the Candy Crush Saga, the three-dimensional huge urban space in the Assassin's Creed and the Call of Duty, and even the two-dimensional fine structure in the Angry Birds and the open world based on voxel in the Minecraft. At the same time, PCG's commercial applications already exist in the industry, including Rogue and the Dark Destroyer series inspired by Rogue, and more recently Civilization IV and Minecraft.

At present, the research hotspot of PCG of levels and maps is how to generate diversified or adaptive content in a completely autonomous [4] or mixed-dominant manner [5]. Content generation technology and content evaluation are two key points. In terms of generation technology, the main methods currently include technologies based on search, solver, grammar, cellular automata, fractal and deep learning. Recently, some people have also tried to generate game content through reinforcement learning. Due to the complexity and subjectivity of the group structure of game players and the quality of content may be affected by algorithms and their implied randomness, the evaluation of generated content is a challenging process at present.

This paper focuses on the procedural content generation of game levels and maps, aiming to summarize previous studies. At present, in the field of game levels and maps generation, there are few reviews based on the classification of game types. Therefore, this paper introduces the current research methods according to game types, hoping to combine the existing reviews in the current field to have good inspiration for readers.

II. PROCEDURAL GENERATION METHODS

This paper introduces three classic game types of levels and maps generation methods, including racing games, platform games, and 3D open world games, including a variety of PCG methods. The evaluation methods of generating content are mainly based on representation, agent testing, and player testing.

A. Racing Game

Racing Game is a type of electronic game, which mainly competes with the speed of the first person or the third person. The reason for choosing to research racing games is that the game type can be directly mapped to the real world, which is of special significance. Racing games are mainly car racing games, as well as some unconventional flying racing games, science fiction racing games, and special racing games. Taking the car racing game as an example, this paper introduces the procedural content generation of tracks. The current PCG research on tracks is relatively few, and most of the methods are based on search, aiming to pursue diversified, personalized high-quality tracks.

A study [6] has proposed a Cascading Elitism algorithm to generate 2D tracks in pursuit of personalized racetrack generation. The algorithm is essentially a variant of an evolutionary algorithm for multivariate problems. The main content of the algorithm is to sort the population individuals according to the fitness standard defined for each generation, leaving a specified number of individuals, while the others are discarded. If there are other fitness standards, the process is repeated according to the importance of fitness, and then the final surviving individuals are used to supplement the population to the initial number through mutation. In the whole process, crossover variation is not used. The track is

encoded as the sequence of fixed-length fragments. The length of each fragment is fixed and limited to three curvatures. Each fragment is mutated from a certain probability to other types of the fragment. The author defines three fitness degrees including progress difference (the difference between the actual completion distance and the target distance of the racing controller), maximum speed and total progress variance (the track with high progress variance is more challenging). The racing controller is trained according to the player's performance, and finally the controller can finish the journey close to the person used for training on the same track and in the same period. Then the performance profile of the controller on the track is used as fitness to screen the track population, and finally the track that meets the player's technical level is obtained. The result is shown in Fig. 1.

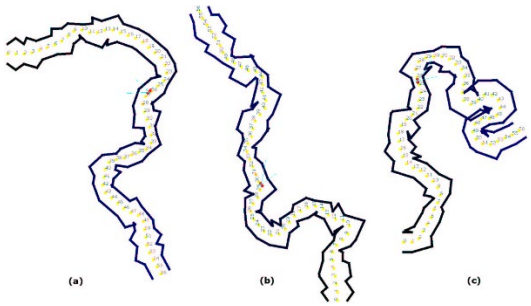

Fig. 1. Three evolved tracks: (a) evolved for a bad player with target progress 1. 1, (b) evolved for a good player with target fitness 1. 5, (c) evolved for a good player with target progress 1. 5 using only progress fitness [6].

Reference [7] has also used the Cascade Elitism algorithm to generate 2D tracks, and further pursued personalization based on reference [6]. Based on the racing controller in Reference [6], the authors improve it to enable it to imitate the driving style of players, such as running in the middle of the straight road but running near the inner wall in the smooth curved road. And the authors change the fitness ranking, followed by progress difference, total progress variance, maximum speed. Each track segment is represented by two control points. The complete track contains 30 track segments. The mutation is realized by perturbation of the position of the control points. The experiment explores the effects of different track coding representations and different mutation operations on the generation of tracks. The first method represents the track as a set of points in the space. The initial population includes the round-angle rectangular track and the track generated by the round-angle rectangular track after the random walk, and the Gauss mutation method is used to disturb the coordinates of points. The second method represents the track as a group of points with radial configuration under polar coordinates, and the initial population is a circular track. The mutation randomly changes the distance between the point and the track center, and the angle position remains unchanged.

Cardamone et al. [8] have used interactive evolution to generate 3D track in Torcs environment. The authors build an online platform in the experiment. Users can participate in the process of interactive evolution to generate the track through the functions provided by the system. There are two modes of track evolution: single-user mode and multi-user mode. In the single-user mode, the user fully controls the evolution process and defines the fitness of the track by scoring 1-5 points for the track or choosing whether to like (like corresponding 5 points, do not like the corresponding 1 points). In multi-user mode, each user participates in scoring for the track, and the track fitness depends on the scoring average. In the evolutionary process, the racetrack genotype is represented as a set of control points in polar coordinates, and the phenotype is an ordered list of segments in the Torcs environment. The experiment shows that this interactive evolutionary strategy can improve user satisfaction and generate a track that meets the subjective wishes of players.

Loiacono et al. [9] have extended the radial coding method of track and combined information entropy to generate diversified tracks. The authors discuss the relationship between the curvature distribution, velocity distribution and the diversity of the track, and put forward the information entropy of curvature distribution and velocity distribution as two fitness measures of the track. The authors discuss the evolution results of the track by maximizing the curvature distribution entropy and the velocity distribution entropy, respectively, and the evolution of the track by maximizing the two at the same time. The genotype of the track is expressed as a set of control points in polar coordinates, and the phenotype is a second-order Bessel curves sequence. The experiment also verifies the influence of different numbers of control points on the track generation and tests whether the defined fitness is consistent with the player's experience.

A study [10] has proposed technology for real-time adjustment of game experience in games. Combining the gameplay data from the game and the data provided by the sensor, the relevant features are extracted through machine learning technology to construct the player model, and then the next stage of the track is generated according to the player' s performance. Building a player model includes the following processes. Feature extraction, that is, extracting features from user input, game output, eye tracker, and head pose of corresponding track segments from raw data provided by game APIs and sensors. Calculating the performance target, that is, the state of the shortest time of the road section is set to the best player state, and the characteristics of the best player state on the track segment are used as the best performance target. Once the player's time on the track segment is shorter, the best performance target will be updated. Building a weight model, that is, calculate the weight of each feature in each track segment. Then combined with the theoretical framework of behavioral analysis, the feature subset and three high-level aspects of the user model are corresponding, namely experience, exploration and physiological attention; Finally, combined with the theory of flow, the player's demand for each section is judged, including maintaining the same, easier or more challenging track. The track is represented as a sequence of track segments, each segment in the algorithm is represented as a nine-order Bessel curve. When a simple segment is needed, calculate the mean path the player has taken through that segment before. This mean path serves as the optimal path to the center of the new segment that is created. If the player needs a more challenging segment, the difficulty of the road is increased by increasing the angle of the curve or the number of turns.

B. Platform Game

Platform Game is a sub-category of action games. Representative works such as Super Mario Bros. The main game mode is to move and pass-through various obstacles on the suspended platform in various ways on the 2D plane. The game's environment is usually set with uneven terrain. To cross them, players need to manipulate the characters to jump

or climb between these terrains. This article will take Super Mario Bros. as an example to introduce the PCG of the level of the platform game. At present, there are many PCG methods at the Super Mario Bros. level, and this paper summarizes seven methods.

Reference [11] has generated levels based on Multi-dimensional Markov chains (MdMCs) [12]. Generate levels in three ways. (1) Generating and testing, that is, simply regenerate the whole content until the required constraints are met. (2) Discovering and regenerating the part of the illegal constraint. (3) Incremental method, that is, generate and check the level part. The level of Super Mario Bros. is represented as a two-dimensional array, each cell corresponds to a tile, and sentinel elements are added to the left and bottom of the level. The constraints include aesthetic constraints and playability constraints. Aesthetic constraints mainly refer to no damaged pipes. Playability constraints include playability, number of pipes, number of gaps, and the maximum length of gaps. But not all the three generation methods support all constraints. The experiment lists the constraint subsets that each method matches.

Hauck et al. [13] have proposed a levels generation system based on graph grammar. The system can extract data from the input map, process the data, and finally recombine new levels. The level is represented as a graph. Each game tile is associated with the nodes in the graph, and the edges of the nodes are connected to its adjacent tile nodes. As shown in Fig. 2, the algorithm includes three processes: structural identification, structural matching and level generation. There are three inputs in the structure recognition phase, including a series of levels, the minimum number of structures to be identified and the maximum edge length of the structure. The method is to select as far as possible equidistant n non-air tiles in the level, create nodes for these tiles, indicating that the tile is in (x, y) coordinates, and then expand outward from these nodes at the same time. The end condition of the expansion is to collide with other structures or reach the maximum edge length. After the end of the expansion, a connector node is created for each structure and is placed in the (x, y) position of the opposite collision structure node. The structure matching stage evaluates which institutions can be connected through the structure consistency and accessibility, in which the structure consistency is determined by the location of the connector, and the accessibility is verified by the reachability concept [14], forming a list of each structure that can be connected to other structures, and the probability distribution can be set. In the level generation phase, generate new levels based on the syntax obtained in the previous two phases. There are two constraints in this phase. The first is the availability of connector nodes (ensure at least one entry point for substitution at every step of the generation). Second, preventing the overlap of a new joining structure with a structure joined at a previous step of the generation. For the second constraint, the authors discuss whether the overlap of air tiles is regarded as an illegal constraint and analyze the experimental results.

Green et al. [15] have used the FI-2Pop [16] evolutionary algorithm to focus on the generation of playable levels similar to input levels in the pass action sequence. Mario agent will trigger actions when playing games, such as jumping, eating gold coins and so on. The genotype of the level is encoded as the action sequence corresponding to the contained scene. The experiment uses both crossover and mutation for level evolution. Crossover allows the exchange of any number of scenes between checkpoints. Mutation refers to the operation of adding or deleting the scene and changing the structure of the scene. Playability is guaranteed by the threshold of average completion of the agent after running N times on the level. Playability depends on the average completion degree of the agent after running N times on the level. The similarity is measured by comparing the error between the input action sequence and the agent's actual action sequence. Errors include missing action and redundant action. The fitness function is proposed from two aspects of errors, and finally generates playable levels similar to the input level in the sequence of the pass actions.

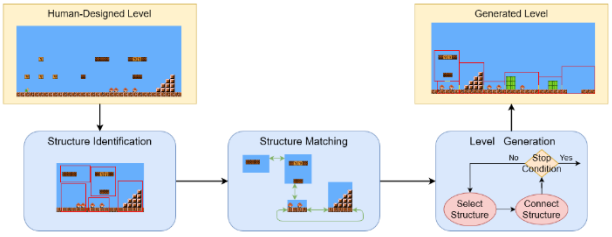

Fig. 2. The three main stages of the proposed system [13].

Summerville et al. [17] used Long Short-Term Memory networks (LSTMs) [18] to generate Mario levels. As shown in Fig. 3, the input layer of the network is composed of one-hot vector encoding, and each kind of tile has a unique encoding. The authors consider three factors in the generation process, including the route to generate the tiles: bottom-to-top or snaking generation, whether the training set contains the player's pass path, and whether column depth information is considered in the training process. The authors train the generator networks for each of the eight scenarios and test their prediction accuracy on the test set. Among them, The Snaking-Path-Depth has the lowest error. At the same time, the experiment also shows that including the player's pass path in the training set can improve the playability of the generated levels.

In Reference [19], the idea of the LVE algorithm [20] was introduced into the generation of levels, and CMA-ES [21] was used to explore the potential vector space in the GAN network [22] to generate the levels with the specified attribute target. As shown in Fig. 4, the method is divided into two stages. Firstly, the GAN network is trained using the existing Mario levels, which are encoded into two-dimensional arrays. After the generation network training is completed, the exploration process of the potential space is placed under the CMA-ES evolutionary control, and the fitness function based on the representation, and the fitness function based on the agent test are used to generate levels with specific properties.

Fontaine et al. [23] have used the most advanced quality diversity algorithm (QD) [24] to explore the potential vector space of GAN (DCGAN) and generated a high-quality set of levels with different eigenvalues of specified feature dimensions. The authors use three quality diversity algorithms include MAP-Elite [25], MAP-Elite (line) [26] and CMA-ME [27], Random and CMA-ES [28] methods to explore the potential vector space. The generation targets are set based on representation, agent test, KL divergence [29], and the generators are evaluated for different targets. The evaluation indexes include the percentage of playable cards, expression range, QD-Score [30], and more. The experimental results show that QD algorithms MAP-Elite, MAP-Elite (line) and

CMA-ME are superior to CMA-ES and random search in finding high-quality scenarios with specified attribute dimensions. In addition, CMA-ME is superior to other test algorithms in the diversity and quality of generated scenes, as is shown in Fig. 5.

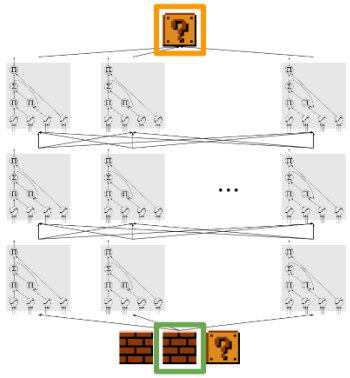

Fig. 3. Graphical depiction of our chosen architecture. The green box at the bottom represents the current tile in the sequence, the tile to the left the preceding tile, and the tile to the right the next tile. The tile in the top orange box represents the maximum likelihood prediction of the system. Each of the three layers consists of 512 LSTM blocks. All layers are fully connected to the next layer [17].

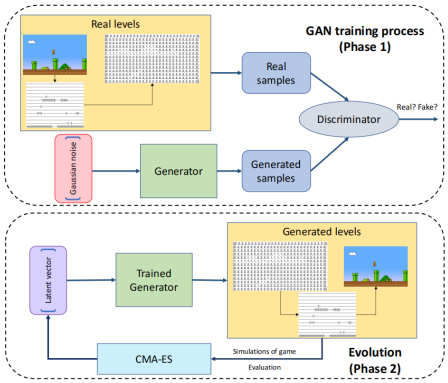

Fig. 4. Overview of the GAN training process and the evolution of latent vectors [19].

Sarkar et al. [31] have used VAE (CVAE) to generate Mario levels. The experiment uses binary vector coding conditional labels to control the game elements and design patterns contained in the generated results. For game elements, the length of the conditional label vector is 5, which corresponds to five game elements: enemy, pipeline, coin, fragile brick and question mark brick, respectively. 0 / 1 represents the absence/existence of game elements. For SMB design patterns, the authors pick 10 such patterns based on the 23 described by Dahlskog and Togelius (2012), such as Enemy Horde (EH): a group of 2 or more enemies, Gap (G): 1 or more gaps in the ground. Each input data involved in training CVAE is associated with a label vector. Then the potential variables of random sampling and the associated conditional labels are input into the trained decoder, which can realize the controllability of generating level fragments. This method can also be used to mix different platform games.

Open World Game

Open World game, also known as Free Roam, is a kind of game level designed in which players can freely roam in a virtual world and freely choose the time and way to complete the game task. This section will introduce the application of PCG in open world maps from three aspects: terrain, architecture and cave.

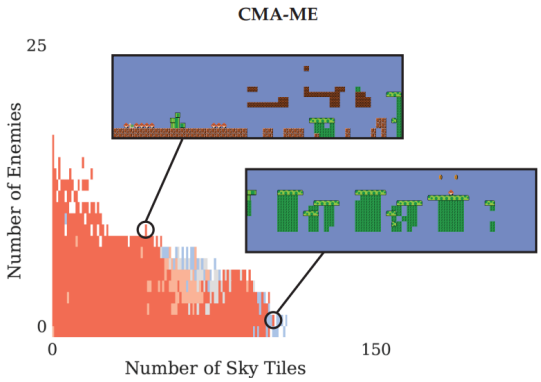

Fig. 5. Mario scenes returned by the CMA-ME quality diversity algorithm, as they cover the designer-specified space of two level mechanics: number of enemies and number of tiles above a given height. The color shows the percentage of the level completed by an A* agent, with red indicating full completion [23].

Frade et al. [33] have proposed a new technique, Genetic Terrain Programming based on the evolutionary design with GP allow game designers to evolve terrains according to their aesthetic feelings and desired features. The developed application produces Terrains Programs that will always generate different terrains, but consistently with the same features (e. g. valleys, lakes). The terrain is represented as a heightmap. Each GP individual is a tree composed of mathematical functions and heightmaps as terminals. Some terminals depend upon a Random Ephemeral Constant (REC) to define some characteristics, such as inclinations of planes and sizes of the geometric figures terminals. All these terminals depend upon a random number generator, which means that consecutive calls of one TP will always generate different terrains. The population size of evolution and when to stop evolution can be artificially controlled. The initial population is randomly generated, and the designer can perform mutation and crossover operations. The individual leaving completely depends on the designer. The generator can eventually generate aesthetically attractive terrain and terrain with specific features.

Reference [34] has proposed a method of building generation using architectural profiles based on answer set programming. The architectural profile is a kind of semantic specification, which restricts a building solver in a declarative way. The method extends the framework of general building solvers. The general building solver considers tiles, adjacent conditions between building tiles, and validity constraints. The method proposed by the author extends the framework of the general building solver: the products of the general building solver are taken as stage products, and these products are defined as shapes, which can be further combined into complex buildings. Compared with the general building solver, the method in this paper pays more attention to the placement of shapes in the input space. The architectural profiles consist of five parts. (1) Tiles: Each tile has architectural meaning, such as walls, windows and doors. (2) Adjacency Conditions: Semantics is also defined on the adjacencies between tiles, expressing the meaning of how tiles may relate to each other. (3) Shapes: The central concept of an architectural profile is a shape, defined as a connected cluster of tiles. (4) Shape

adjacency conditions: If two tiles in two shapes can be connected, then the two shapes can also be connected. (5) Architectural validity constraints: it can help steer the generation process towards the domain of plausible and feasible architectural models through traversability, gravity, and density constraints. Finally, the architectural profiles are transformed into logical constraints, and the model is generated by the solver.

Freiknecht et al. [35] have presented a program generation method for multistory buildings that contain stairwells and can be traversed. Each building contains a set of rooms connected by doors or stairs and is equipped with windows. The building generation process is divided into object model construction and 3D model generation. Object model building includes the following steps. (1) Design the building shape. (2) Choose the stairwell type. (3) Place the stairwell: Determine the location of stairwells by the optimal fitting algorithm in the vertical shared area of all floors. (4) Place rooms: The algorithm automatically places enough initial rooms with a reasonable layout at each floor shape vertex or subdivision vertex. (5) Expand rooms: Loop to expand each initial room to determine the final size of each room. (6) Find the longest corridor: Find the longest path connecting with the stairwell and connecting all rooms. (7) Then extend the path to a polygon. If the corridor does not need to connect the exterior wall, delete the last edge of the polygon in turn until the deletion will reduce the number of connected rooms and reach the final state. (8) Merge corridors and stairwells. (9) Add Windows and doors: According to the topology of the building, the location, size and adjacency of the room, the building adds the outer door, the inner door, the outer window and the inner window in turn. (10) Calculate the roof area: Calculate the roof area for each floor of the building and specify the type of roof for each roof. The 3D model generation process converts the room information, stair information and roof information contained in the object model into a building model with specified quality. In this process, the author modifies the index group of common grid structures to promote the texture of the grid.

Mark et al. [36] have proposed a modular pipeline to generate underground caves, which includes structural components, cave generator and renderer. Structural components use the L-System to generate a set of overall structural points of the cave. In this process, the authors reduce the self-similarity of the generation structure by longer production rules and fewer rewritings, and introduce the ability of randomly generating production rules to improve the expression ability of the generator. At the same time, the random process is constrained to retain the reliability of the generation process. Then, a certain proportion of dead alleys are linked by curves, and the tree structure is changed into the cave structure. The cave generator uses a twisted metaball to move at the structural point of the cave to build a real cave tunnel. Then, based on the noise value computed for each voxel, placement points are found for stalactites and stalagmites, and objects are grown at these locations by cellular automata. The renderer converts the volume of the cave voxels into 3D geometric data by the Marching Cube algorithm, then calculates the grid normal by sampling the density value of the adjacent voxels, and smoothes the surface, and then applies the texture to the UV-free program grid by using three-sided projection to produce results comparable to those in the real world. At the same time, it can adjust the parameters of the shader and modify the style of the 3D cave greatly.

III. Conclusion

This paper summarizes sixteen methods of generating levels and maps for racing games, platform games and open world games, and describes the ideas and central processes of these methods. Here is a summary of the main content.

A. Racing Game

In this paper, the method of generating racing game levels is mainly for the track generation of car racing games. At present, there is not much research in this area. The generation method is mainly based on search, and the core is the evolutionary algorithm. Personalized tracks are generated by combining direct player modeling or indirect player modeling, and diversified tracks are generated by combining information entropy. This paper thinks that more research should be devoted to the generation of the track because car racing games can directly map to real activities. Optimizing players' behavior in the game through a personalized track can also promote real driving behavior. In terms of track generation technology, we think it can be developed from three aspects. First, the track height variable should be considered in the generation process because the track height gap is an important factor for a good experience. Second, there are a lot of tracks in the real world, and we can develop a tool to restore the track according to the track image. Thirdly, more accurate track quality evaluation methods should be found from different perspectives.

B. Platform Game

The methods for the generation of platform game checkpoints are aimed at the generation of Super Mario Bros. levels in this paper. The seven methods are based on search, graph grammar, and machine learning methods to generate playable, like the input levels or generate diverse levels. Most of these methods have a certain degree of versatility in the generation of platform game levels. From the trend point of view, the generation method gradually moves closer to deep learning. But deep learning doesn't seem to capture the functional needs and aesthetic attributes of a level very well. Although some studies have considered the constraints of playability and aesthetic attributes, the generated levels still cannot meet the commercial standards, and there are still unplayable parts or wrong tiles in the levels. Therefore, this paper argues that after the generation of levels, it should also go through a repair link to repair the functional and aesthetic attributes of customs and promote its commercialization process. However, there are few studies on this point and need to be studied.

C. Open World Game

About the method of open world game maps generation, this paper introduces the terrain, architecture and cave generation to supplement the integrity of the article. The generation method is mainly based on search, fractal, solver and cellular automata. Although it is not detailed in the description of this paper, the generation of open world game maps is also a field with great potential.

Finally, the following is the understanding of the PCG process of game levels and maps. (1) First, we must determine the basic objectives of the content we want to generate, and this paper argues that includes three aspects. The first is the pursuit of high-quality generated content. In this process, the versatility of the method is not considered, and the game

content is devoted to generating extreme aesthetics and meeting functional requirements. The second is to pursue the general method of generating the content, including two aspects. To introduce the method ideas of other fields into PCG, or to propose new methods to generate the same type of game content from the game content, such as generating the Mario levels according to the existing Mario levels. Third, look for relationships between different types of game content and generate different types of game content from the existing game content, such as generating game scenarios based on music. (2) According to our basic goal, we determine the representation method of game levels and maps. In this process, if we refer to the construction method of the mapping object of the game content in the real world, it may benefit to construct a more appropriate representation method. (3) Based on the first two stages, it can be combined with other fields such as player modeling to achieve a fully autonomous or mixed-dominant game content generator for a diversified or personalized generation. Among them, the diversification goals include generating controllable attribute levels, similar levels to input ones, and the interpolation expansion of discretely distributed levels to make them relatively continuous levels space. There are two kinds of personalized performance. One is real-time adjustment according to the player's performance in the game round. The other is to provide users with appropriate levels and maps outside the round based on diversity generation. In the future, we believe that PCG can be widely used in game development in industry, and industry and academia will work together to promote the development of PCG, and create high-quality games in which all aspects of game content are interrelated procedurally. Moreover, PCG can be combined with the Metaverse to shine.

ACKNOWLEDGMENT

This paper is supported by the Fundamental Research Funds for the Central Universities under Grant Number: N2017003.

REFERENCES

[1] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural content generation in games*. Cham, Switzerland: Springer International Publishing, 2016.

[2] A. Liapis, G. N. Yannakakis, and J. Togelius, "Computational game creativity," in ICCC., 2014.

[3] G. N. Yannakakis and J. Togelius, *Artificial intelligence and games*, vol. 3. New York, NY, USA: Springer, 2018, pp. 184-193.

[4] G. N. Yannakakis, "Game AI revisited," in *Proc. 9th Conf. Comput. Frontiers.*, 2012, pp. 285-292.

[5] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," in *ACII.*, 2015, pp. 519-525.

[6] J. Togelius, R. De Nardi and S. M. Lucas, "Making racing fun through player modeling and track evolution," 2006.

[7] J. Togelius, R. De Nardi and S. M. Lucas, "Towards automatic personalised content creation for racing games," in *IEEE Symp. Comput. Intell. Games.*, 2007, pp. 252-259.

[8] L. Cardamone, D. Loiacono and P. L. Lanzi, "Interactive evolution for the procedural generation of tracks in a high-end racing game," in *Proc. 13th Annu. Conf. Genetic. Evol. Comput.*, 2011, pp. 395-402.

[9] D. Loiacono, L. Cardamone and P. L. Lanzi, "Automatic track generation for high-end racing games using evolutionary computation," *IEEE Trans. Comput. Intell. AI. Games.*, vol. 11, no. 3, pp. 245-259. 2011.

[10] T. Georgiou and Y. Demiris, "Personalised track design in car racing games," in *IEEE CIG.*, 2016, pp. 1-8.

[11] S. Snodgrass and S. Ontanón, "Controllable Procedural Content Generation via Constrained Multi-Dimensional Markov Chain Sampling," in *IJCAI.*, 2016, pp. 780-786.

[12] S. Snodgrass and S. Ontañón, "Experiments in map generation using Markov chains," in *FDG.*, 2014.

[13] E. Hauck and C. Aranha, "Automatic Generation of Super Mario Levels via Graph Grammars," in *IEEE CoG.*, 2020, pp. 297-304.

[14] S. Londoño and O. Missura, "Graph Grammars for Super Mario Bros Levels," in *FDG.*, 2015.

[15] M. C. Green, L. Mugrai, A. Khalifa and J. Togelius, "Mario level generation from mechanics using scene stitching," in *IEEE CoG.*, 2020, pp. 49-56.

[16] S. O. Kimbrough, G. J. Koehler, M. Lu and D. H. Wood, "On a feasible–infeasible two-population (fi-2pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch," *Eur. J. Oper. Res.*, vol. 190, no. 2, pp. 310-327. 2008.

[17] A. Summerville and M. Mateas, "Super mario as a string: Platformer level generation via lstms," unpublished.

[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput*. vol. 9, no. 8, pp. 1735-1780. 1997.

[19] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith and S. Risi, "Evolving mario levels in the latent space of a deep convolutional generative adversarial network," in *Proc. Genetic. Evol. Comput. Conf.*, 2018, pp. 221-228.

[20] P. Bontrager, J. Togelius and N. Memon, "Deepmasterprint: Generating fingerprints for presentation attacks," unpublished.

[21] N. Hansen, S. D. Müller and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)," *Evol. Comput.*, vol. 11, no. 1, pp.1-18. 2003.

[22] I. Goodfellow, J. Pouget-Abadie, M. Mirza and others, "Generative Adversarial Nets," in *Adv. Neural Inf. Process. Syst*., 2014, pp. 2672–2680.

[23] M. C. Fontaine, R. Liu, J. Togelius and A. K. Hoover and S. Nikolaidis, "Illuminating mario scenes in the latent space of a generative adversarial network," unpublished.

[24] J. K. Pugh, L. B. Soros and K. O. Stanley, "Quality diversity: A new frontier for evolutionary computation," *Frontiers Robot. AI*., vol. 3, pp: 40. 2016.

[25] J. B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," unpublished.

[26] V. Vassiliades and J. B. Mouret, "Discovering the Elite Hypervolume by Leveraging Interspecies Correlation," in *Proc. Genetic. Evol. Comput. Conf*., 2018, pp:149–156.

[27] M. C. Fontaine, J. Togelius, S. Nikolaidis and A. K. Hoover, "Covariance matrix adaptation for the rapid illumination of behavior space," in *Proc. Genet. Evol. Comput. Conf*., 2020, pp. 94-102.

[28] N. Hansen, "The CMA evolution strategy: A tutorial," unpublished.

[29] S. M. Lucas and V. Volz, "Tile pattern kl-divergence for analysing and evolving game levels," in *Proc. Genetic. Evol. Comput. Conf.*, 2019, pp. 170-178.

[30] J. K. Pugh, L. B. Soros, P. A. Szerlip and K. O. Stanley, "Confronting the challenge of quality diversity," in *Proc. Annu. Conf. Genetic. Evol. Comput.*, 2015, pp. 967-974.

[31] A. Sarkar, Z. Yang and S. Cooper, "Conditional Level Generation and Game Blending," unpublished.

[32] K. Sohn, H. Lee and X. Yan, "Learning structured output representation using deep conditional generative models," *Adv. Neural Inf. Proc. Syst.*, vol. 28, pp: 3483-3491. 2015.

[33] M. Frade, F. F. De Vega and C. Cotta, "Modelling video games' landscapes by means of genetic terrain programming-a new approach for improving users' experience," in *Workshops Appl. Evol. Comput*., 2008, pp. 485-490.

[34] L. van Aanholt and R. Bidarra, "Declarative procedural generation of architecture with semantic architectural profiles," in *IEEE CoG.*, 2020, pp. 351-358. IEEE.

[35] J. Freiknecht and W. Effelsberg, "Procedural Generation of Multistory Buildings with Interior," *IEEE Trans. Games.*, vol. 12, no. 3, pp: 323-336. 2019.

[36] B. Mark, T. Berechet, T. Mahlmann and J. Togelius, "Procedural Generation of 3D Caves for Games on the GPU," in *FDG*., 2015.