Automatic Generation of Game Levels Based on Controllable Wave Function Collapse Algorithm

Darui Cheng, Honglei Han, Guangzheng Fei

Automatic Generation of Game Levels Based on Controllable Wave Function Collapse Algorithm

Darui Cheng[0000-0003-2910-7437], Honglei Han*[0000-0001-6630-7580], and Guangzheng Fei

Communication University of China, Beijing, China

{diary,hanhonglei,gzfei}@cuc.edu.cn

Abstract. Procedural content generation automatically creates game content through methods such as pseudo-random numbers, which helps save labor or create games that can be played repeatedly and indefinitely. The wave function collapse algorithm is an effective procedural content generation algorithm newly proposed in recent years, but it has the problems of complicated rule writing and lack of non-local constraints. In this paper, based on the original wave function collapse algorithm, an automatic rule system is proposed, which can simplify the rule writing. Moreover, we use the three mechanisms, namely global constraint, multi-layer generation, and distance constraint, to establish non-local constraints. Through experiments, compared with the original wave function collapse algorithm, the results show that manual control has been enhanced, and the generated levels have a certain degree of similarity with human-designed levels. By making a real-time dynamic level game demo using this method, it turns out that the controllable wave function collapse algorithm we proposed has great potential in the game level generation field.

Keywords: Procedural Content Generation, Wave Function Collapse, Level Generation, Game Design.

1 Introduction

Procedural Content Generation (PCG) is a significant research field in game development. To automatically create game content, pseudo-random numbers and other methods have been adopted in PCG fields. The application of PCG in the field of game development helps to save manpower, assist game designers to exert their creativity, save storage space, and create games with infinite replayability[1]. PCG algorithm can be divided into the following four categories: constructive methods, search-based methods, constraint-based methods, and machine-learning methods[2]. Among them, the Wave Function Collapse (WFC) algorithm is a newly proposed PCG algorithm based on constraint solving, which has great potential value in the game development field.

However, there are four problems related to the WFC algorithm: complicated rules configuration, lack of global control, difficulty to achieve distance constraints, and impossible to generate multi-layer levels. Therefore, we propose a method that adds

more constraints to the original WFC algorithm to solves these problems. First, for the complex adjacency rules which need to be manually configured in the original WFC algorithm, we proposed an automatic rule system to reduce the number of rules that need to be written. Second, we added global control in the original WFC algorithm through the global maximum constraints and tiles preset, which provides more capabilities for game designers to control generated results. Then, the distance constraints were integrated into the observation stage of the WFC algorithm. At last, multi-layer levels generation was supported so that game elements with different semantics, such as NPC and interactive items, can be placed on different layers, and there are connections and constraints between each layer.

2 Related Work

2.1 Procedural Content Generation

Previous studies have recognized the importance of PCG in the creation of video games in different aspects. The current use of PCG technology in game development is mainly limited to specific types of game elements. Because the design ideas for different types of game levels are not the same, it is challenging to generate the entire game level procedurally. Lawrence Johnson et al. used the idea of the constructive PCG algorithm to create the infinite cave level. The algorithm based on cellular automata is evaluated in the infinite cave game to generate playable and well-designed cave game levels[3]. Miguel Frade et al. used a genetic algorithm to evolve the terrain to obtain the required game levels with sufficient accessibility[4], which is a search-based algorithm. The PCG via machine learning (PCGML) has also become a research hotspot, and a lot of research has been carried out in this direction. These researches[5–11] each used different machine learning methods to train on the VGLC[12], a small dataset of Super Mario Bros levels, attempting to generate new levels. Nonetheless, none of them meet the requirements of sufficient playability and controllability. Generative Adversarial Networks (GAN) is a machine learning method suitable for generation. Giacomello et al.[13] input a manually extracted level feature vector to the level generator by conditional GAN, trying to generate a level that is similar to human-designed in visual and structural aspects. Volz et al.[14] combined GAN with the evolutionary search. A two-stage experiment tried to use the generated results after screening for repeating train, in order to achieve the directional generation. However, these two studies did not reach the result of the utterly controllable generation of levels.

The mentioned constructed methods, search-based methods, machine learning methods have various drawbacks and are only suitable for specific generation goals. Constructive algorithms use fixed algorithms to generate game content at once without testing and may generate results that are beyond human control as well as unusable[1]. A search-based algorithm specifies an evaluation function and uses the idea of the evolutionary search to obtain a better result through continuous generation and evaluation, whose efficiency is relatively low[15]. All machine learning methods have a common defect that they all need a large dataset for training, but this is not practical

in the field of games. It is difficult to collect a large game level construction dataset manually. Also, different game types require completely different game levels dataset. Each time developing a new game, one needs to reconstruct the dataset. Currently, all PCG methods based on machine learning can only be trained with a very small data set, so that the results obtained generally lack playability, and the failure rate is relatively high. Compared with the above three types of algorithms, the constraint solving algorithm has certain advantages in the generation of game levels, which is specifically reflected in the fact that this type of algorithm can randomly generate levels that fully meet the artificial constraints, and the controllability is higher.

2.2 Constraint Solving Algorithm

Constraint solving algorithm is neither a constructive PCG algorithm nor a strictly search-based PCG method. As an algorithm in the field of traditional artificial intelligence, constraint solving uses ideas from knowledge representation and search to model continuous and combinatorial search and optimization problems and solve them with domain-independent algorithms[16]. The problem of game level generation under certain constraints is actually a constraint satisfaction problem (CSP), which is usually defined according to a series of decisive variables and values. The method of PCG based on constraint solving can usually guarantee the output precisely but at the cost of the unpredictability of the total running time[17].

Michael Cerny Green et al. took the lead in applying the constraint solving algorithm to the generation of dungeons in games. They divided the algorithm into two stages, namely Layout Creator and Game Element Furnisher, using the algorithm based on constraint solving at all of them[18]. However, their method is only applicable to the generation of dungeon rooms or cave levels. It is not possible to control the reasonable adjacency of each tile in the map in more detail, so it is not suitable for outdoor terrain generation. Moreover, because the use of traditional constraint solving methods needs to iterate step by step, a scan has to be run in order to determine the appropriate position when adding each element. So, when there are too many game elements, it is easy to cause confusion and low efficiency.

2.3 Wave Function Collapse Algorithm

WFC algorithm divides game levels into a grid and defines all cells in the grid that have all possible values firstly, reference the concept of superposition state in quantum physics. During the operation, it determines the values in each cell one by one through "observing", and then propagates the effects to adjacent cells, and finally achieve the goal of determining a feasible solution that meets all constraints. Although this algorithm has a probability of failure caused by conflict, the efficiency is higher than the traditional search-based constraint solving method.

Many researchers have applied the WFC algorithm in different studies. Isaac Karth et al. summarized a data-driven branch in WFC algorithms, namely the overlapping model. This model uses the bitmap file as input, directly reads the pixel-to-pixel adjacency relationship in the bitmap file as a constraint, and is used to guide the propaga-

tion stage of the WFC algorithm[17]. The WFC algorithm of the overlapping model has remarkable advantages when used as a texture generator but is not suitable when generating game levels. Werner Gaisbauer et al. tried to generate a virtual city under a specific game theme by adjusting the parameters used in the basic WFC algorithm[19]. They only tried to find the best parameters for the city generation in a specific game, but make no algorithm-level improvements. Hugo Scurti et al. applied the WFC algorithm to generate random paths for NPCs in games[20]. Hwanhee Kim et al. extended the WFC algorithm from the existing grid-based system to the graph-based system, breaking through the limitation of the grid in the original WFC algorithm, making the WFC algorithm more widely applicable[21]. Nevertheless, the above studies have ignored the possibility of integrating non-local constraints and multi-layer generation into the WFC algorithm. In this view, the motivation for our work is addressing this problem.

WFC, an algorithm based on adjacency constraints, lacks non-local constraints in nature. The method described herein combining the original WFC algorithm Maxim Gumin proposed[22] and the dungeon room generator Green et al. presented[18], using the idea of solving CSP, provides a robust control handle for game designers. It can solve problems mentioned above of the WFC algorithm, which is conducive to obtaining a level generation result that is more controllable and more similar to the artificial design.

3 Method

WFC algorithm can be divided into the overlapping model and the simple tiled model. The overlapping model is suitable for procedural texture generation[17]. The simple tiled model can manually configure more complex rules to restrict the adjacency between tiles with different semantics, which is more suitable for the game levels generation. Since the problem we need to solve is the generation of game levels, the simple tiled model is selected.

The method in this paper is shown in Figure 1 and consists of the following four steps, which contains the key improvements that we introduced to the original WFC algorithm.

1. Rule initialization: In the original WFC algorithm, all allowed adjacency rules in the grid were explicitly written into a configuration file. According to the rule library specified in this file, each time the WFC algorithm runs, it comes up with a solution that meets all the adjacency rules, that is, a game level with certain playability. To reduce the number of rules that need to be manually written, we integrate an automated rule system during the rule initialization phase, which can expand one manual rule into multiple equivalent rules based on the symmetry of tiles.
2. Data initialization: Before each generation, we have to clear and initialize all data in the grid. Before the first observation stage, we insert a global minimum constraint stage to fix some preset tiles at specific positions to enhance the game designer's control of the result.

3. Observation: In terms of information entropy theory, we calculate the information entropy H of all cells with equation (1), where P_i means the probability of each available tile. A cell with the lowest information entropy is selected as the observation cell. From all the remaining available tiles in the cell, we select one randomly according to the weight. After the value of a cell is determined, global maximum constraint, inter-layer constraint, and the distance constraint are added to solve a series of problems of the original WFC algorithm.

$$H = -\sum_{i=1}^{n} P_i log P_i \quad (1)$$

4. Propagation: The constraint caused by the value determined in the observation stage is propagated to adjacent cells at the propagation stage. Then, the undetermined cell of the lowest information entropy is selected for further observation. This process is repeated until all cells converge to certain observing values.

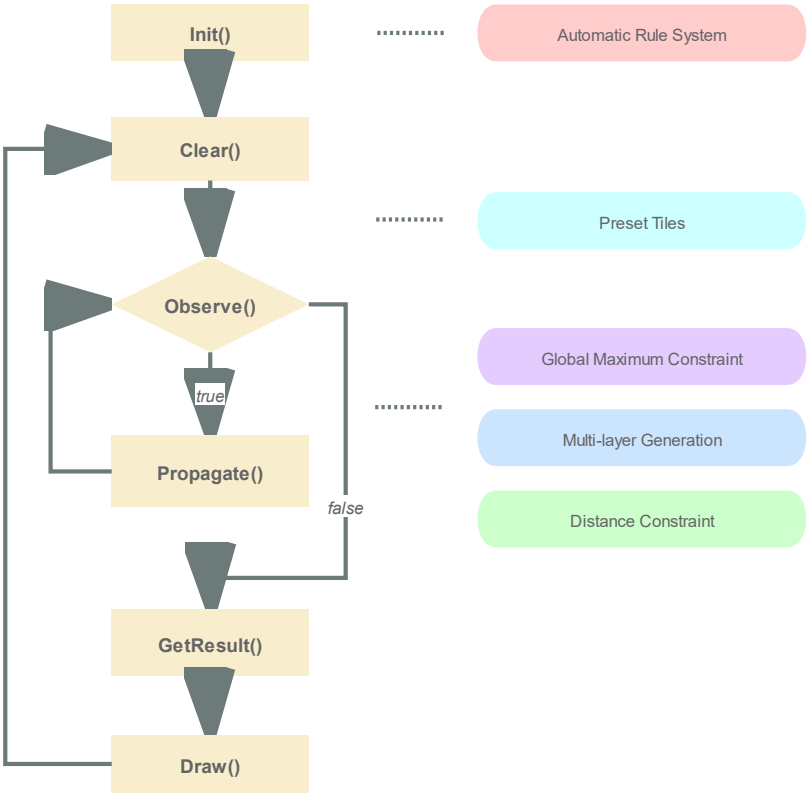


Fig. 1. Main program flow.

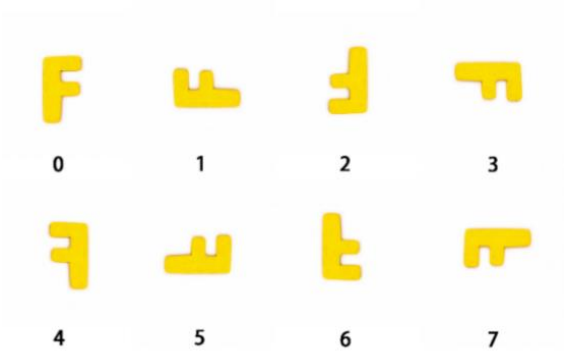


Fig. 2. 8 transformation forms of a tile. The first line is obtained by rotating the No. 0 pattern 90 degrees counterclockwise sequentially, and the second line is left-right mirror images of the first line.

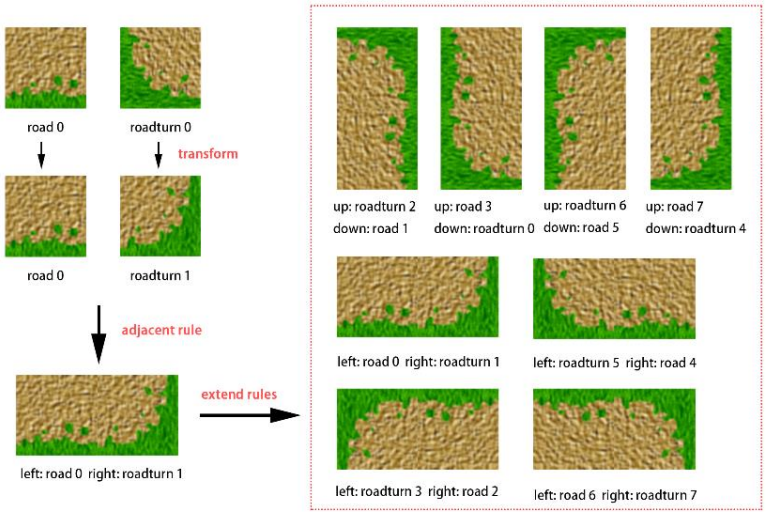


Fig. 3. Get 8 equivalent rules from one left-right rule.

3.1 Automatic Rule System

In order to use transformations of the tile to simplify the rule configuration, as shown in Figure 2, we have summarized 8 commonly used transformation forms of a tile, and each is marked with an index number, which is convenient for search.

First, we temporarily ignore the symmetry of tiles and expand each rule. As shown in Figure 3, the No. 0 transformation of road tiles and No. 1 transformation of road turn tiles form an adjacency rule based on manual configuration. By rotation and mirroring, this rule can be expanded into 8 rules. Since the rules between the two tiles are mutual, the rule is expanded into 16 rules for storage finally.

Next, we consider the symmetry of the tiles. The method mentioned above can solve the problem of repeating different transformation forms of the same rule, but there are still too many rules to write. For rules with symmetrical tiles, the equivalent rules can be extended using symmetry to reduce the number of the required rules further. Therefore, on the basis of the original WFC algorithm[22] and the WFC algorithm of Arunpreet Sandhu et al.[23], the common symmetry types of tiles need to be extended. To build a more comprehensive system of symmetry types of tiles, as shown in Table 1, the symmetry types of tiles are summarized into nine types. The types are named by a letter with the same symmetry type.

8 transformation forms and 9 symmetry types will form a complicated mapping relationship. How to quickly find the transformation number of the equivalent tile after the symmetry transformation of the tile becomes the key point to the automatic rule system. For this reason, we have proposed the concept of the symmetry dictionary, which is established by recording the transformation number equivalent to the initial tile of each symmetry type. The symmetry dictionary is shown in Table 1.

The automatic rule system expands manually configured rules into a large number of equivalent rules, eliminating the game designer's workload of configuring these equivalent rules, which significantly simplifies the difficulty of writing the configuration file. For example, if all 4 sides of 8 transformation types of 2 tiles need to be configured to allow adjacency, 8×8×4×2 = 512 rules have to be manually written. With the automatic rule system, 1 to 8 rules are simply needed, then other rules are extended by this system. As a result, the efficiency of rules writing has been improved significantly.

Table 1. Symmetry dictionary.

Name	Symmetry type	Initial tile's equivalent transformation number	New symmetry type after transformation
F	No symmetry	0	F/F/F/F/F/F/F/F
S	Centrosymmetric	0/2	S/S/S/S/S/S/S/S
T	Vertical axis symmetry	0/4	T/B/T/B/T/B/T/B
L	Counter-diagonal axis symmetry	0/5	L/Q/L/Q/Q/L/Q/L
B	Horizontal axis symmetry	0/6	B/T/B/T/B/T/B/T
Q	Main diagonal axis symmetry	0/7	Q/L/Q/L/L/Q/L/Q
I	Horizontal and vertical axis symmetry	0/2/4/6	I/I/I/I/I/I/I/I
/	Double diagonal axis symmetry	0/2/5/7	'/'/'/'/'/'/'/'/'/'/'/'/'/'
X	All 8 transforms are identical	0/1/2/3/4/5/6/7	X/X/X/X/X/X/X/X

3.2 Non-local Constraints

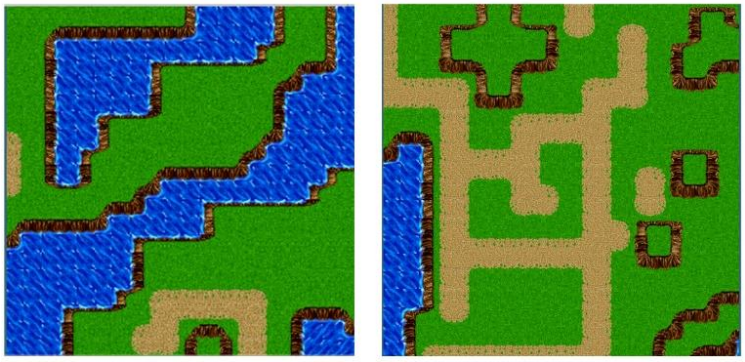
Global Constraint. One of the main problems of the WFC algorithm is the lack of global constraint, which includes two aspects: global maximum constraint and global minimum constraint. Global maximum constraint refers to the limitation of the maximum number of a specific kind of tile, while the global minimum constraint is that a specific type of tile appears at least several times. The lack of global constraints will

cause the global number of game elements generated in the level not to be controlled by the WFC algorithm.

The effect of applying the global maximum constraint is shown in Figure 4. The constraint is not added in Figure 4(a), which results in the water area is too large. In Figure 4(b), the global maximum constraint of the water tiles is set to 20 units and get the result we expect.

The global minimum constraint can be equivalent to presetting the minimum number of tiles at specified positions or random positions at first. The method of presetting tiles is to set the value of the specified cell as the specified tile after the data initialization stage and then to spread its influence. Game designers can also use this function to presets the outstanding design that they need to set in advance, and the WFC algorithm automatically generates the remaining parts.

The effects of the global minimum constraint are shown in Figure 5. Due to the position (0, 0) is preset to water, as shown in the result of Figure 5, it can be seen that the WFC algorithm generates the rest of the level without violating the preset constraints.



(a) (b)

Fig. 4. Two generated levels without(a) and with(b) global maximum constraint.

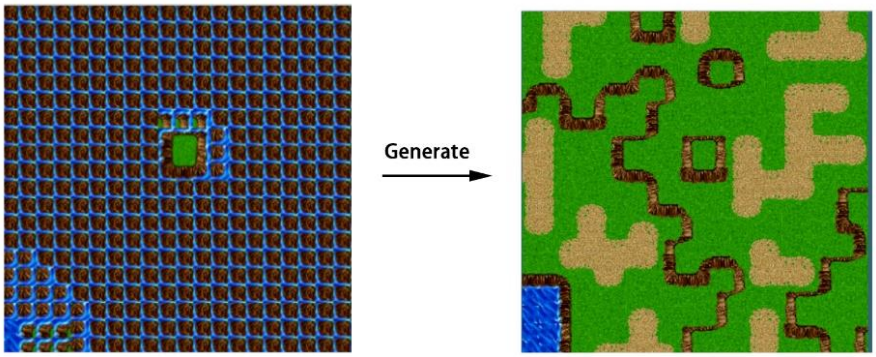
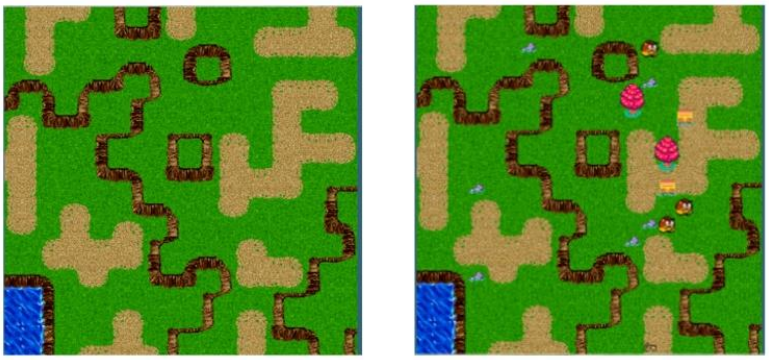


Fig. 5. Global minimum constraint effects.

Multi-layer Generation. A game scene is often composed of multiple layers of game elements with different semantics and requires different types of game elements to be generated at different layers. Because game elements of different layers can overlap each other, they usually do not apply to adjacency constraint rules but require inter-layer constraint. It is difficult to achieve with the original WFC algorithm. Figure 6(a) is the result of generating a level of a single layer, which generally can only be applied to simple types of game scenes. However, in Figure 6(b), the enemy and the treasure chest are added, the enemy can only be generated on the grass, and the treasure chest can only be generated on the road. It is a result of using the method of this study.

To achieve multi-layer generation, after each observation of the bottom layer is completed, the tiles which do not meet the inter-layer constraints in the upper cell are banned. In this way, through the definition of the inter-layer constraint rules, the connection between the two layers is established.

With the function of multi-layer generation, the richness and playability of the game levels have been improved. The automatically generated game levels are no longer just a flat one-layer game map.



(a) (b)

Fig. 6. Two generated levels, with single-layer generation(a) and double-layer generation(b).

Distance Constraint. The distance between the game elements is closely related to the playability of the game, so it is a significant factor for game designers to build game scenes. However, the WFC algorithm is mainly built on the adjacency constraints and lacked the constraints of distance, which makes some game design relying on distance constraints cannot be achieved. As shown in Figure 7(a), the distance between the treasure chest and the enemy cannot be effectively constrained, which makes it impossible to achieve the design expectations that the enemy surrounds the treasure chest.

In view of this, it is necessary to control the maximum and minimum distance between tiles and combine adjacent constraints to achieve the effect of comprehensive distance constraints.

In Figure 7(b), when game designers limit the distance between treasure chests and enemies less than 10 units and limit the distance between treasure chests and keys greater than 10 units, the generation results can meet the design goals of enemies distributed around the treasure chest and the keys kept away from the treasure chest.

(a) (b)

Fig. 7. Two generated levels, without(a) and with(b) distance constraints.

4 Experiments and Analysis

All experiments in this study were completed on a computer with a CPU of Intel Core i7-8750H and a graphics card of Intel UHD Graphics 630.

4.1 Comparison with the Original WFC Algorithm

First, to verify the improvement of this method after adding more constraints on the original WFC algorithm, an experiment was conducted to compare the results of this method with the original WFC algorithm when generating a birds-eye-view 2D game level. According to a game designer's requirements, the automatically generated level needs to meet the following three constraint rules:

1. Global Constraint. In the case of a scene size of 20×20, the maximum number of water tiles is not more than 20 units. Besides, the bottom left corner (0,0) floor tile is water, and the center (10,10) tile is grass.
2. Distance constraint. The minimum distance between a treasure chest and a key is 10 units, and the maximum distance is 20 units. The maximum distance between a treasure chest and an enemy is 10 units.
3. Inter-layer constraint. Trees, rocks, and enemies must be generated on the grass, and treasure chests and keys must be generated on the road.

The final result is shown in Figure 8, the two sub-pictures in (a) use Maxim Gumin's WFC algorithm level generator [22], and the two sub-pictures in (b) use the method of our study. Due to the original WFC algorithm cannot generate multiple

layers, the results of the original algorithm do not generate the second layer of elements. In the original WFC algorithm, as shown in Figure 8(a), the global parameters are not controlled, only the visually reasonably map generated. In particular, with the large water area as well as the lack of decorations and interactive items, the map can barely present playability enough. Although a nice visual effect has been achieved, it is actually not suitable for the game generation. In Figure 8(b), the global maximum constraint was used in our study to limit the water area to less than 20 units. At the same time, the multi-layer generation method was used to generate double layers, so that more playable elements are generated above the map layer. The two preset tiles are marked with blue squares in Figure 8(b), and the number of water tiles does not exceed the maximum constraint of 20 units so that the game designer achieve the goal to generate a small pool in the lower-left corner. The distance between the treasure chest and the key is marked with an orange circle in Figure 8(b), which can be seen that meets the constraint of 10-20 units. The light-yellow rectangular frame in Figure 8(b) shows the aggregation effect of the distance constraint, which makes enemies always appear around the treasure chest. Finally, we can find trees, rocks, monsters always generated over grass, while treasure chests and keys are always generated on the road, which meets the strict restriction rules between layers.

(a) (b)

Fig. 8. Comparison between method in study[22](a) and our method(b).

4.2 Similarity to Human-designed Levels

To analyze the distinction between the game levels we generate automatically and the levels created by the game designers manually, we conducted a user survey. We invit-

ed 5 game designers to design 5 levels manually and generated 5 levels by our method automatically under the same constraints, then mixed them and shuffled the order to form a questionnaire. Five game designers were invited to ensure that the survey results are not affected by the personal design style of the game designer. The questionnaire informed the respondents of the underlying meaning of the game elements and the design intention in these levels. Meanwhile, we informed them that there were only 5 human-designed levels. Then the respondents were required to choose 5 levels which they think are manually designed from 10 game levels. Finally, we got 151 valid feedbacks from students with some game experience, and the statistical result is shown in Table 2.

From Table 2, it can be seen that the game levels generated by the algorithm in this paper are similar to the game levels designed by humans. Ideally, 50% levels that were generated automatically should be considered as human-designed, so do the indeed human-designed levels, meaning that the respondents cannot distinguish the generated level at all. In the worst case, 0% levels that were generated automatically should be considered as human-designed, while the human-designed levels can be 100% identified, meaning that the respondents completely distinguish the generated levels. The survey results show that 57.62% of the respondents think that the No. 1 generated level is human-designed, which has exceeded 50%. The percentage of generated levels considered to be human-designed reaches an average of 40.13%, which is close to 50%. It shows that most participants were failed to differentiate the design made automatically or manually.

Table 2. The user survey results.

Level number	Manual 1	Manual 2	Manual 3	Manual 4	Manual 5	Average
Percentage considered to be human-designed	61.59%	32.45%	70.20%	66.23%	68.87%	59.87%
Level number	Auto 1	Auto 2	Auto 3	Auto 4	Auto 5	Average
Percentage considered to be human-designed	57.62%	40.40%	37.75%	33.11%	31.79%	40.13%

4.3 Real-time Dynamic Level Generation

Finally, we used the controllable WFC algorithm proposed in this paper to implement a real-time game demo that dynamically generates game levels at runtime. It is a 3D top-down roguelike shooter game with rich terrain with plateaus, brooks, and sinuous roads, whose goal is to defeat the enemies and collect treasures from chests. In this game, all levels are not generated before the game starts, but are generated during the player's game. In this way, it is possible to read the player's operation data in real-time during the game and modify the constraint configuration as a parameter, and then dynamically change the difficulty or style of the game level generated later depend on the different performance of players. For example, when the player defeats a lot of enemies, we can increase the difficulty of subsequent levels; when the player takes a

large amount of damage from enemies, we will generate easier levels later on. The real-time level generation effect is shown in the attached video.

5 Conclusion and Future Work

This paper mainly solves two main problems of the WFC algorithm: the complex constraint rules are hard to write; excessively random game levels that lack control are always less playable. We proposed automatic rule system, global constraint, multi-layer generation, distance constraint to solve these problems effectively and provide a controllable game level generation tool for the game designers. Experiments show that the method is better than the original WFC algorithm in the aspects of enhancing the designer's control and generation effect. Moreover, the game levels automatically generated by this method achieve a high similarity with the human-designed game levels. In addition, Real-time dynamic level generation experiment shows that the method in this paper can be successfully applied to practical games generation and get satisfactory results.

This method is only applicable to some game types whose game scenes are built on a plane and is hard to used directly into some other game types such as 2D side-scrolling games. In future work, we plan to extend this method to the field of 2D side-scrolling game levels generation. Furthermore, we will also do more research on the real-time generation of levels by this controllable WFC algorithm.

Acknowledgments. This work was supported by the Fundamental Research Funds for the Central Universities, and the National Key R&D Program of China (2018YFB1403900).

References

1. Shaker, N., Togelius, J., Nelson, M.J.: Procedural Content Generation in Games. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-42716-4.
2. Summerville, A., Snodgrass, S., Guzdial, M., Holmgard, C., Hoover, A.K., Isaksen, A., Nealen, A., Togelius, J.: Procedural Content Generation via Machine Learning (PCGML). IEEE Transactions on Games. 10, 257–270 (2018). https://doi.org/10.1109/TG.2018.2846639.
3. Johnson, L., Yannakakis, G.N., Togelius, J.: Cellular automata for real-time generation of infinite cave levels. In: Proceedings of the 2010 Workshop on Procedural Content Generation in Games - PCGames '10. pp. 1–4. ACM Press, Monterey, California (2010). https://doi.org/10.1145/1814256.1814266.
4. Frade, M., de Vega, F.F., Cotta, C.: Evolution of Artificial Terrains for Video Games Based on Accessibility. In: Di Chio, C., Cagnoni, S., Cotta, C., Ebner, M., Ekárt, A., Esparcia-Alcazar, A.I., Goh, C.-K., Merelo, J.J., Neri, F., Preuß, M., Togelius, J., and Yannakakis, G.N. (eds.) Applications of Evolutionary Computation. pp. 90–99. Springer Berlin Heidelberg, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12239-2_10.
5. Dahlskog, S., Togelius, J., Nelson, M.J.: Linear levels through n-grams. In: Proceedings of the 18th International Academic MindTrek Conference on Media Business, Management, Content & Services - AcademicMindTrek '14. pp. 200–206. ACM Press, Tampere, Finland (2014). https://doi.org/10.1145/2676467.2676506.

6. Summerville, A., Philip, S., Mateas, M.: MCMCTS PCG 4 SMB: Monte Carlo Tree Search to Guide Platformer Level Generation.
7. Summerville, A., Mateas, M.: Super Mario as a String: Platformer Level Generation Via LSTMs. arXiv:1603.00930 [cs]. (2016).
8. Hoover, A.K., Togelius, J., Yannakis, G.N.: Composing Video Game Levels with Music Metaphors through Functional Scaffolding.
9. Snodgrass, S., Ontanon, S.: Learning to Generate Video Game Maps Using Markov Models. IEEE Trans. Comput. Intell. AI Games. 9, 410–422 (2017). https://doi.org/10.1109/TCIAIG.2016.2623560.
10. Jain, R., Isaksen, A., Holmga, C., Togelius, J.: Autoencoders for Level Generation, Repair, and Recognition.
11. Guzdial, M., Riedl, M.: Learning to Blend Computer Game Levels. arXiv:1603.02738 [cs]. (2016).
12. Summerville, A.J., Snodgrass, S., Mateas, M., Ontañón, S.: The VGLC: The Video Game Level Corpus. arXiv:1606.07487 [cs]. (2016).
13. Giacomello, E., Lanzi, P.L., Loiacono, D.: Searching the Latent Space of a Generative Adversarial Network to Generate DOOM Levels. In: 2019 IEEE Conference on Games (CoG). pp. 1–8. IEEE, London, United Kingdom (2019). https://doi.org/10.1109/CIG.2019.8848011.
14. Volz, V., Schrum, J., Liu, J., Lucas, S.M., Smith, A., Risi, S.: Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network. arXiv:1805.00728 [cs]. (2018).
15. Togelius, J., Yannakakis, G.N., Stanley, K.O., Browne, C.: Search-Based Procedural Content Generation: A Taxonomy and Survey. IEEE Transactions on Computational Intelligence and AI in Games. 3, 172–186 (2011). https://doi.org/10.1109/TCIAIG.2011.2148116.
16. Russell, S.J., Norvig, P.: Artificial intelligence: a modern approach. Prentice Hall, Englewood Cliffs, N.J (1995).
17. Karth, I., Smith, A.M.: WaveFunctionCollapse is constraint solving in the wild. In: Proceedings of the International Conference on the Foundations of Digital Games - FDG '17. pp. 1–10. ACM Press, Hyannis, Massachusetts (2017). https://doi.org/10.1145/3102071.3110566.
18. Green, M.C., Khalifa, A., Alsoughayer, A., Surana, D., Liapis, A., Togelius, J.: Two-step Constructive Approaches for Dungeon Generation. arXiv:1906.04660 [cs]. (2019).
19. Gaisbauer, W., Raffe, W.L., Garcia, J.A., Hlavacs, H.: Procedural Generation of Video Game Cities for Specific Video Game Genres Using WaveFunctionCollapse (WFC). In: Extended Abstracts of the Annual Symposium on Computer-Human Interaction in Play Companion Extended Abstracts - CHI PLAY '19 Extended Abstracts. pp. 397–404. ACM Press, Barcelona, Spain (2019). https://doi.org/10.1145/3341215.3356255.
20. Scurti, H., Verbrugge, C.: Generating Paths with WFC. arXiv:1808.04317 [cs]. (2018).
21. Kim, H., Lee, S., Lee, H., Hahn, T., Kang, S.: Automatic Generation of Game Content using a Graph-based Wave Function Collapse Algorithm. In: 2019 IEEE Conference on Games (CoG). pp. 1–4. IEEE, London, United Kingdom (2019). https://doi.org/10.1109/CIG.2019.8848019.
22. Gumin, M.: Bitmap & tilemap generation from a single example by collapsing a wave function, https://github.com/mxgmn/WaveFunctionCollapse.
23. Sandhu, A., Chen, Z., McCoy, J.: Enhancing wave function collapse with design-level constraints. In: Proceedings of the 14th International Conference on the Foundations of Digital Games - FDG '19. pp. 1–9. ACM Press, San Luis Obispo, California (2019). https://doi.org/10.1145/3337722.3337752.