UDC 004.94

Kodola G.N., Denysiuk O.R., Khrebet M.A.

ABOUT PROCEDURAL GENERATION OF THE CONTENT AND ITS USE AT THE CREATION OF COMPUTER GAMES

Ukrainian State University of Chemical Technology, Dnipro

Procedural generation of content is crucial when creating various elements of computer games, such as game levels, buildings, textures, weapons, faces of non-player characters, trees, bushes, etc. Procedural generation is understood as a process of automatic creation of various parts of a game by mixing various input data. In the paper this process is demonstrated by the example of modeling of a room with different dimensions and different number of outputs. In case of proper implementation, it is possible to base more than 90% of a game on procedural generation of content, which saves a significant amount of memory used for content storage. The paper presents a brief overview of the first systems of game generation, their development path from participation in the creation of small levels to participation in the creation of whole games. One of the first games based on the use of procedural generation was Rogue, which spawned a whole genre called roguelike. The paper considers examples of the successful application of various procedural generation algorithms in games of different genres such as: complex dynamic relationships in Crusader Kings II; unique enemies in Shadow of Mordor; procedural generation of the world in Civilization. Examples of use of procedural generation in other spheres, such as modeling 3D objects, or work of artists, are given. The algorithm for procedural generation of the game level is developed. A level is constructed from the cells depending on various options for constructing the level – with clear and free boundaries, the formation of the cells with possible output options, constructing their set and implementing the passages between neighboring cells. The implementation of the proposed algorithm for procedural generation of a level built on cells will allow creating a universal program for the formation of various levels (maps) and can be further integrated into the game, which saves considerable time and resources.

Keywords: procedural content generation, computer game, algorithm, game level, universal game programs.

Problem statement

In the modern gaming industry procedural generation is used so often that its various manifestations can be seen in almost every game of any genre. Procedural generation is an algorithm that can create game content independently and without direct human involvement. In simple terms, procedural generation is a process of creating something by mixing various input data in different ways. As a result of this mixing, with a proper variety of input data, we can get a unique product.

The procedural generation can be widely used in the process of game development, for example when creating:

- game levels;
- buildings;
- textures;
- weapon;
- faces of non-player characters;
- trees, bushes, etc.

The implementation of algorithms for procedural content generation has practical importance, since the solution of this task will not only significantly diversify computer games by creating completely new game mechanisms and genres, but also significantly save the amount of memory used for content storage.

Analysis of recent research and publications

Research in the field of universal game programs was primarily focused on the study of artificial intelligence algorithms, and the generation of games either was not considered in these works at all, or

was based on the simplest algorithms.

The works of C. Browne [1], V. Hom and J. Marks [2], J. Togelius [3], M. Cook [4] are devoted to research on procedural generation of games. Existing experimental solutions are focused on the generation of games of several separate genres.

A promising direction for further research is to improve existing and create new generation algorithms in order to improve quality and variety of generated games.

Formulation of the research objective

The objective of the paper if to perform a review and analysis of research on the procedural generation of content and its use in the creation of computer games, to propose an algorithm for procedural generation of a game level.

Exposition of the main material of the study

With proper implementation, more than 90% of the entire game can be built on procedural content generation [5].

To understand what procedural generation is, consider a simple example. As such, let us examine the algorithm for the procedural generation of a small room.

There are input data – the size of a room and the number of doors. After this, a procedural generation algorithm is developed, which will randomly mix these two parameters. As a result, after starting the generation algorithm, one of the possible variants of room design is obtained as an output. For example, if the room size can be 20 m^2, 30 m^2 or 40 m^2, and the number of doors can be 1, 2 or 3, the output of the algorithm will be one of 9 possible options.

Thus, the programmer, using a small amount of input data, can generate a huge number of different unique rooms. 9 possible options from the example were obtained only with the use of 2 input parameters, and in case of full algorithm of procedural generation that can have more than 100 different incoming parameters, each of which has the same number of possible values, the number of possible outputs can be enormous. That's why procedural generation has earned itself such a popularity among ordinary players and developers who create games.

The undoubted advantage of procedural content generation is the amount of memory used. In the example above, storage of all 9 possible rooms in memory will take, for example, 90 kilobytes. But the storage of the algorithm for procedural generation of these rooms in the memory can take around 20 kilobytes. Thus, we can say that the use of procedural generation algorithms perfectly saves device memory [6].

Development of procedural generation algorithms

The first widely known applications of procedural generation date back to the early 1980s, when with limited resources of computers it was already possible to create large and diverse worlds. Typical examples are Rogue and Elite.

Rogue, in turn, became so popular that it has spawned an whole genre based on procedural generation used in it. Subsequently, this genre would be called Roguelike. An integral part of any roguelike game is the presence of randomly generated levels, which players will explore.

It has already been a long time since the early 1980"s, and during this time, procedural generation algorithms have come a long way from participating in the creation of small levels to participating in the creation of whole games. So, due to the development of procedural generation, such games as Diablo, Spore, Borderlands, Don't Starve and many others were created [7,8].

After release of Rogue, other games of this type began to appear much later, but procedural generation in them has also reached a completely different level. For example, in Diablo, released in 1996, the mechanic of procedural generation was used to create levels, enemies and even weapons.

Another excellent example of introducing procedural generation into the game is Spore, released in 2008. In this game, literally everything is created based on the mechanics of procedural generation – from the creation of non-game creatures, to the buildings where they live and the entire their planets.

A newer example of such a game is No Man's Sky, where an entire galaxy is created by procedural generation algorithms, with unique planets and animals to be explored by the player.

As a result, it can be said that games completely based on the mechanics of procedural generation appear more often, and their scales are increasing [8].

Algorithms for procedural generation are fundamentally different, depending on an area in which they are used. For example, in computer games algorithms can be responsible for the generation of game levels (Diablo, Heroes of Might and Magic), the generation of random weapons (Diablo, Borderlands), names of non-player characters and enemies (Diablo, S.T.A.L.K.E.R.).

Some examples of the successful application of various procedural generation algorithms in games of different genres are given below:

1. Complex dynamic family ties in Crusader Kings II:

With the help of procedural generation, characters of the strategy Crusader Kings II set in

the Middle Ages become unique, though computer-controlled, personalities. Various traits, such as intrigue or greed, passing through the algorithm of procedural generation, determine the future behavior of a character and his relationship with others.

2. Unique enemies in Shadow of Mordor:

In this game, based on The Lord of the Rings by J.R.R. Tolkien, the main enemy for the player are orcs, created using procedural generation. Using the Nemesis System algorithm, when creating a new enemy, data such as name, appearance, manner of speaking and even relationships with other orcs is taken into account. Combining so many different parameters allows the game to create more and more new combinations of enemies with their own characteristics. Also, after participating in battles, some orcs may obtain procedurally generated scars, or they may be promoted by awarding a new rank. All this creates the impression that the player is surrounded by unique, detailed enemies.

3. Procedural generation of the world in Civilization:

The use of procedural generation in creation of new maps allows the developers to significantly diversify the game process, because each subsequent map is not similar to the previous one – it encourages players to explore a new game space, and also does not let the game to become boring.

However, procedural generation has found its use not only in games. For one thing, it is often used as a tool in modeling. An example is a simple algorithm for procedural generation of a 3D tree model. In this algorithm, a stem is taken as the basis of a future tree, to which, as the algorithm progresses, different, procedurally created branches are attached, and the leaves, in turn, are attached to those branches. Such an algorithm allows designers to save their time significantly, as well as to achieve the unlikeness of all trees, as it happens with trees in real life [9,10].

Artists who work on creating different textures also appreciate procedural generation. There are two ways to create textures – creating a texture manually or using photos as a basis for a future texture. An artist can also combine these two methods. Both methods are very labor-intensive, because they require a lot of time and effort to provide the final version. On the other hand, with the help of algorithms of procedural generation it is possible to create textures, which almost do not require any finalization. And when creating similar textures, it's enough to just replace several parameters of input data in the algorithm. An artist can not do this using any other method.

Development of the procedural generation algorithm

For clarity, we can consider a simple algorithm for procedural generation of a game level, which is constructed from cells [11,12].

First you need to decide how the final level will look. There are two possible options for its design – with clear or free boundaries. They are provided in Figures 1 and 2, respectively.

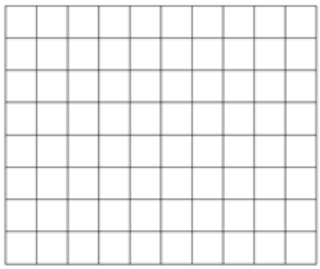

Fig. 1. Level design with clear boundaries

The option with clear level boundaries assumes much less variation in the level design, and therefore the second option is chosen.

After the style of level design is chosen, it is necessary to decide whether cells of a level will be different, or will they be flat fields. Since the given example can then be used in a game, the first option is selected. On the stage of the development of cells, the question of their implementation arises. There are two possible options:

– to create a set of cells that will be randomly selected and set to the level;

– to write a separate algorithm of procedural cell generation for the level.

The second option assumes greater uniqueness

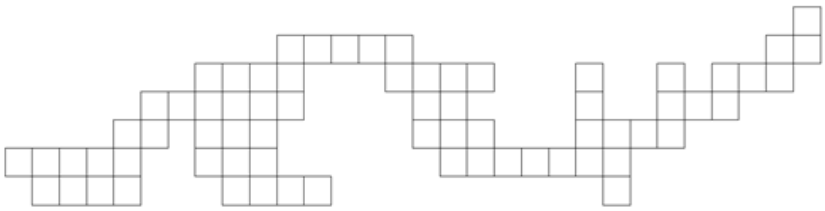

Fig. 2. Level design with free boundaries

of each of the created cells, and therefore the level as a whole, but it requires a lot of effort to implement, and therefore the choice falls on the first option. Then it is necessary to create a set of cells, from which the level will be built.

If there are too few created cells, they will often repeat at the level. It is also crucial to figure out how to implement a system of passes between neighboring cells. Again, there are two solutions:

– to create a large number of cells with all possible variants of passages, so that each cell can be placed in a specific place, depending on the connection with other cells;

– to create several cells, and to write a function that will rotate the cells and change the number of passages depending on where the cell needs to be placed.

If we use the first method, we will have to manually create many unique cells: 4 cells with one passage, 2 cells with I-shaped passages (Fig. 3), 4 cells with L-shaped passages (Fig. 4), 4 cells with T-shaped passages (Fig. 5), and 1 cell with passages on each side of the cell. It turns out that Totally it's necessary to create 15 unique cells, and as a result each turn will look the same as the previous one.

Fig. 3. Two cells with I-shaped passages

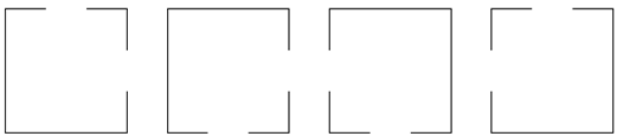

Fig. 4. Four cells with L-shaped passages

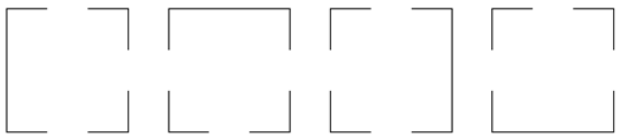

Fig. 5. Four cells with T-shaped passages

The second option looks more attractive, since it is possible to create several cells and a function that will rotate them and change the number of passages. Thus, if you create a cell that has passages on each side, the function will be able to change it and leave, for example, only two of them. This will introduce a small element of randomness into the level, and therefore it is decided to use the second option.

So, after combining all the steps together, we can say that the algorithm for procedural level generation is almost ready. Ehere will be only cosmetic changes and work on possible errors.

Thus, the example of an algorithm for procedural generation of a level built on cells was considered. With further work, this procedural generation algorithm can be integrated into the game.

Conclusions

As it was shown in the paper, procedural content generation has found its niche in many areas of game development. It saves a lot of time and resources. As a result, it can be said that procedural generation has already reached great heights, but still has not reached its limit. And soon enough the procedural generation algorithms will reach such a scale that they will be used not only in games, but also in other spheres of human activity, for example, in education or military affairs. In games, procedural generation has almost reached its peak, because you can already see games that are completely built using procedural mechanics. However, it is still not a limit.

Current paper presents the development of the algorithm for procedural generation of a game level, which is constructed from cells depending on various versions of the level construction – with clear or free boundaries, the formation of cells with possible output options, constructing their set and implementing the passages between neighboring cells. The implementation of the proposed algorithm for procedural generation of a level built on cells will allow creating a universal program for the formation of various levels (maps) and can be further integrated into the game, which saves considerable time and resources.

REFERENCES

1. *Browne C.* Evolutionary Game Design. – Berlin: Springer, 2011. – 122 p.

2. *Hom V., Marks J.* Automatic Design of Balanced Board Games // Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment Conference. – 2007. – P.25-30.

3. *Togelius J., Schmidhuber J.* An Experiment in Automatic Game Design // IEEE Symposium on Computational Intelligence and Games. – 2008. – P.111-118.

4. *Cook M., Colton S.* Multi-Faceted Evolution Of Simple Arcade Games // IEEE Conference on Computational Intelligence and Games. – 2011. – P.289-296.

5. *Shaker N., Togelius J., Nelson Mark J.* Procedural Content Generation in Games. – Springer, 2016. – 218 p.

6. *Texturing* and Modeling: A Procedural Approach, Third Edition / Ebert David S., Musgrave F. Kenton, Peachey D. and etc. – Morgan Kaufmann, 2002. – 688 p.

7. *Hendrikx M., Meijer J.S., Van Der Velden A.I.* Procedural Content Generation for Games: A Survey // ACM Transactions on Multimedia Computing, Communications, and Applications. – Feb. 2013. – Vol.9. – No. 1. – P.1-22.

8. *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. – Sept. 2011. – Vol.3. – No. 3. – P.172-186.

9. *Yannakakis G.N.* Experience-Driven Procedural Content Generation // IEEE Transactions on Affective Computing. – July 2011. – Vol.2. – No. 3. – P.147-161.

10. *Sorenson N.* A Generic Approach to Challenge Modeling for the Procedural Creation of Video Game Levels // IEEE Transactions on Computational Intelligence and AI in Games. – Sept. 2011. –Vol.3. – No. 3. – Pp.229-244.

11. *Martin A.* Evolving 3D Buildings for the Prototype Video Game Subversion // Proceedings of the 2010 International Conference on Applications of Evolutionary Computation. – Nov. 2010. – Vol.6024. – P.111-120.

12. *Ashlock D.* Search-Based Procedural Generation of Maze-Like Levels // IEEE Transactions on Computational Intelligence and AI in Games. – Sept. 2011. – Vol.3. – No. 3. – P.260-273.

Received 14.04.2018

ПРО ПРОЦЕДУРНУ ГЕНЕРАЦІЮ КОНТЕНТУ ТА ЇЇ ВИКОРИСТАННІ ПРИ СТВОРЕННІ КОМП'ЮТЕРНИХ ІГОР

Кодола Г.М., Денисюк О.Р., Хребет М.О.

Процедурна генерація контенту є однією з найбільш актуальною при створенні різних елементів комп'ютерних ігор, таких як ігрові рівні, будівлі, текстури, зброї, осіб неігрових персонажів, дерев, кущів та ін. Під процедурною генерацією розуміють процес автоматичного створення різних складових частин гри шляхом «змішування» між собою різних вхідних даних, що демонструє розглянутий в статті наочний приклад моделювання кімнати різних розмірів з різною кількістю виходів. При правильній реалізації на процедурній генерації контенту може бути побудовано більше 90 % всієї гри, що дозволяє значно заощадити обсяг використовуваної пам'яті для зберігання контенту. У роботі наданий короткий огляд перших систем генерації ігор, їх шлях розвитку від участі в створенні невеликих рівнів, до участі в створенні цілих ігор. Однією з перших ігор заснованій на використанні процедурної генерації була гра Rogue, яка породила цілий жанр, так званий Roguelike, тобто Rogue-подібні. Розглянуто приклади вдалого застосування різних алгоритмів процедурної генерації в іграх різних жанрів, таких як: складні динамічні родинні зв'язки в Crusader Kings II; унікальні вороги в Shadow of Mordor; процедурна генерація світу в Civilization. Наведені приклади використання процедурної генерації в інших сферах, таких як моделювання 3D-об'єктів, робота художників. Надана розробка алгоритму процедурної генерації ігрового рівня, який будується з клітин в залежності від різних варіантів побудови рівня – з чіткими і вільними межами, формування самих клітин з можливими варіантами виходів, побудови їх набору і реалізації проходів між сусідніми клітинами. Реалізація запропонованого алгоритму процедурної генерації рівня, побудованого на клітинах, дозволить створити універсальну програму формування різних рівнів (карт) і може бути в подальшому інтегрована в гру, що дозволяє значно заощадити час і ресурси.

Ключові слова: процедурна генерація контенту, комп'ютерна гра, алгоритм, рівень гри, універсальні ігрові програми.

О ПРОЦЕДУРНОЙ ГЕНЕРАЦИИ КОНТЕНТА И ЕЕ ИСПОЛЬЗОВАНИИ ПРИ СОЗДАНИИ КОМПЬЮТЕРНЫХ ИГР

Кодола Г.Н., Денисюк О.Р., Хребет М.А.

Процедурная генерация контента является одной из наиболее актуальной при создании различных элементов компьютерных игр, таких как игровые уровни, здания, текстуры, оружия, лиц неигровых персонажей, деревьев, кустов и др. Под процедурной генерацией понимают процесс автоматического создания различных составляющих частей игры путем «смешивания» между собой различных входных данных, что демонстрирует рассмотренный в статье наглядный пример моделирования комнаты различных размеров с разным количеством выходов. При правильной реализации на процедурной генерации контента может быть построено более 90% всей игры, что позволяет значительно сэкономить объем используемой памяти для хранения контента. В работе представлен краткий обзор первых систем генерации игр, их путь развития от участия в создании небольших уровней, до участия в создании целых игр. Одной из первых игр основанной на использовании процедурной генерации была игра Rogue, которая породила целый жанр, называемый Roguelike, то есть Rogue-подобные. Рассмотрены примеры удачного применения различных алгоритмов процедурной генерации в играх разных жанров, таких как: сложные динамические родственные связи в Crusader Kings II; уникальные враги в Shadow of Mordor; процедурная генерация мира в Civilization. Приведены примеры использования процедурной генерации в других сферах, таких как моделирование 3D-объектов, работа художников. Представлена разработка алгоритма процедурной генерации игрового уровня, который строится из клеток в зависимости от различных вариантов построения уровня – с четкими и свободными границами, формирования самих клеток с возможными вариантами выходов, построения их набора и реализации проходов между соседними клетками. Реализация предложенного алгоритма процедурной генерации уровня, построенного на клетках, позволит создать универсальную программу формирования различных уровней (карт) и может быть в дальнейшем интегрирована в игру, что позволяет значительно сэкономить время и ресурсы.

Ключевые слова: процедурная генерация контента, компьютерная игра, алгоритм, уровень игры, универсальные игровые программы.

ABOUT PROCEDURAL GENERATION OF THE CONTENT AND ITS USE AT THE CREATION OF COMPUTER GAMES

Kodola G.N., Denysiuk O.R., Khrebet M.A.

Ukrainian State University of Chemical Technology, Dnipro, Ukraine

Procedural generation of content is crucial when creating various elements of computer games, such as game levels, buildings, textures, weapons, faces of non-player characters, trees, bushes, etc. Procedural generation is understood as a process of automatic creation of various parts of a game by mixing various input data. In the paper this process is demonstrated by the example of modeling of a room with different dimensions and different number of outputs. In case of proper implementation, it is possible to base more than 90% of a game on procedural generation of content, which saves a significant amount of memory used for content storage. The paper presents a brief overview of the first systems of game generation, their development path from participation in the creation of small levels to participation in the creation of whole games. One of the first games based on the use of procedural generation was Rogue, which spawned a whole genre called roguelike. The paper considers examples of the successful application of various procedural generation algorithms in games of different genres such as: complex dynamic relationships in Crusader Kings II; unique enemies in Shadow of Mordor; procedural generation of the world in Civilization. Examples of use of procedural generation in other spheres, such as modeling 3D objects, or work of artists, are given. The algorithm for procedural generation of the game level is developed. A level is constructed from the cells depending on various options for constructing the level – with clear and free boundaries, the formation of the cells with possible output options, constructing their set and implementing the passages between neighboring cells. The implementation of the proposed algorithm for procedural generation of a level built on cells will allow creating a universal program for the formation of various levels (maps) and can be further integrated into the game, which saves considerable time and resources.

Keywords: procedural content generation, computer game, algorithm, game level, universal game programs.

REFERENCES

1. Browne C. Evolutionary Game Design. Berlin: Springer, 2011. 122 p.

2. Hom V., Marks J. Automatic Design of Balanced Board Games. Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment Conference, 2007, pp.25-30.

3. Togelius J., Schmidhuber J. An Experiment in Automatic Game Design. IEEE Symposium on Computational Intelligence and Games, 2008, pp.111-118.

4. Cook M., Colton S. Multi-Faceted Evolution Of Simple Arcade Games. IEEE Conference on Computational Intelligence and Games, 2011, pp.289-296.

5. Shaker N., Togelius J., Nelson Mark J. Procedural Content Generation in Games. Springer, 2016. 218 p.

6. Texturing and Modeling: A Procedural Approach, Third Edition. Ebert David S., Musgrave F. Kenton, Peachey D. and etc. Morgan Kaufmann, 2002. 688 p.

7. Hendrikx M., Meijer J. S., Van Der Velden A. I. Procedural Content Generation for Games: A Survey. ACM Transactions on Multimedia Computing, Communications, and Applications, 2013, Vol. 9, No. 1, pp.1-22.

8. 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, 2011, Vol. 3, No. 3, pp.172-186.

9. Yannakakis G.N. Experience-Driven Procedural Content Generation. IEEE Transactions on Affective Computing, 2011, Vol. 2, No. 3, pp.147-161.

10. Sorenson N. A Generic Approach to Challenge Modeling for the Procedural Creation of Video Game Levels. IEEE Transactions on Computational Intelligence and AI in Games, 2011, Vol. 3, No. 3, pp.229-244.

11. Martin A. Evolving 3D Buildings for the Prototype Video Game Subversion. Proceedings of the 2010 International Conference on Applications of Evolutionary Computation, 2010, Vol. 6024, pp.111-120.

12. Ashlock D. Search-Based Procedural Generation of Maze-Like Levels. IEEE Transactions on Computational Intelligence and AI in Games, 2011, Vol. 3, No. 3, pp.260-273.