Procedural Content Generation in Game Development Process

Grzegorz Jaśkiewicz

Warsaw University of Technology, The Faculty of Electronics and Information Technology ul. Nowowiejska 15/19, 00-665 Warsaw, Poland

This article describes procedural content generation algorithms used by an independent video game developer in a level design process for the logical game *Keri Tap*. Genetic algorithms were used as the computational core of the level generation routines. The research that was carried out in order to define good algorithm setup has been described. Main idea of this article is to show that PCG [14] methods can be used by small independent video game developers to gain measurable benefits.

Keywords: Procedural content generation, Game development, Video games, Genetic algorithms, Level design.

Background

A video game consists of *a game engine* and *digital content*. A game engine is a software component which simulates and visualizes a game world. It handles graphical rendering, player input, network communication, etc. Digital content is a form of description of a game world — it contains graphics, models, animations, maps, level scripts, dialogues, etc. Comparing the most important equation of theoretical computer science [16]:

programs = algorithms + data structures

the game engine serves as *algorithms* component and the digital content as the *data structures* part. Whereas the game engine can be bought as *off the shelf* or *configure to order* product, it always takes much effort to create digital content, if the game is meant to be unique. Production of digital content sometimes consumes even 90% of the game development effort, whereas rest is spent on customizing the game engine and crafting supporting tools.

Therefore any savings in the content part of the game are very beneficial for the game producer. Some producers provide players user-friendly development tools and encourage them to create own content like game levels, own avatars or game objects. With such means game producers are trying to utilize power of Web 2.0 [12] trend. This could make games more interesting and also give players more content to consume [11]. According to the *90-9-1* rule [10] only small fraction of active players generates most of game content, so to take the advantage of the player-generated content a game must have a huge community of active players. Also a special software infrastructure is needed to run this process. These two factors generally makes player-generated content out of the reach for small developers launching a new game.

Other approach is a Procedural Content Generation (PCG) [7]. In such approach, instead of using human labour, computers are employed to generate game content. PCG is methodology which includes wide family of algorithms and techniques. Particular PCG solutions are tailored to concrete use cases. In some cases there exists ready-made solutions which are only tuned up and used in game, but it is also common for game developers to develop own PCG solutions. The choice of particular solution depends on many factors, which include type of asset to be generated, specific asset requirements as well as execution time given for algorithm to generate an asset. The content is generated either in a game design phase or in a gameplay.

The moment when PCG can take place has significant impact on time execution constraints for PCG. An example of PCG that takes part in a gameplay is dynamically building dialogs with the game characters by constructing sentences using knowledge and grammatical rules rather than static dialog trees. Open worlds [8] in games are implemented with help of the ingame PCG. An example of PCG in the design phase is creation of 3D models of trees using tree growth schema [13] rather than just work of a graphic designer. PCG in a gameplay should not be computationally intensive in order not to impede the game performance, while on PCG used in a design phase more resources could be spent. Additionally, the results of the design phase PCG could be reviewed and enhanced as necessary.

The next section of the article describes development and application of PCG method in a logic game *Keri Tap*. This is an interesting case study, because game was developed by small game developer and using PCG has impacted positively much of the production phase. Discussed PCG algorithm was developed by the game developer.

Methods and Materials

Keri Tap was manufactured by an independent game developer *Crazy Rabbit Lab sp. j.* The game was build and sold for iOS devices. The game consisted of 30 puzzles. Each puzzle was played on a rectangular grid with the square fields. Some fields were inactive. There were also some player pawns on the board. The goal of the player was to visit all the active fields with any pawn. A visited field turned inactive after visiting it. Some fields were special e.g. they could be entered and left from the specified direction. The sample puzzle taken out from the game has been shown in the Fig 1.

The project team consisted of 4 people, who shared responsibilities of a programmer, graphical artist, sound artist, level designer, tester and sales person[1]. For independent video game developers small teams are very common.

The level design process assumed that there will be 60 levels which could be reviewed, rated by difficulty and the half of them would be included in the game distribution. The task of the level design turned out to be tedious and in the long run levels created by one person turned out to be repetitive. In order to aid level designer there was created a program which generated puzzles with solutions. Then the level designer could just review and enhance them.

The program for puzzle generation has created random boards by starting with a complete board and selecting at random fields which were turned inactive — occasionally some symmetries were applied to obtain more eye-catching board. Then the program tried to select starting locations for pawns in order to minimalize

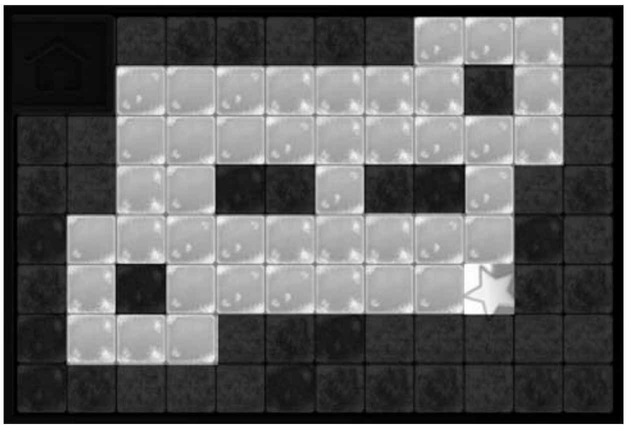

Fig. 1. Sample puzzle in Keri Tap. Dark squares are inactive fields, light squares are active fields. The star acts as a pawn — it stands on the starting field.

the number of pawns on the board that are still able to solve the puzzle. In order to achieve this, the program has searched for the minimal Hamiltonian path coverage [1] of the board represented as a sub-graph of a rectangular lattice [3]. There was a constraint put on paths which disallowed to cross each other. The start of each path indicated to the starting position of the pawn. Each path has described the possible trajectory for the pawn to solve the puzzle. The sample path coverage was shown in the Fig. 2.

The genetic algorithm [6] was used to iteratively improve the set of path coverages for the given graph and to find the best solution. PCG method were applied earlier to similar problems e.g [4] and there existed successful applications of GA in PCG e.g in [9]. However graph theory related concept of the *Keri Tap* game was not widely discussed and new algorithm setup had to be defined and tested. Path coverages were treated as alleles in this algorithm. In order to do so, they were encoded as permutations of the board fields. If consecutive fields in the permutation were adjacent on the board, they were treated as a part of the same path. Otherwise, the second field was treated as a start of the new path. For example one of possible alleles for the path coverage presented in figure (2) is:

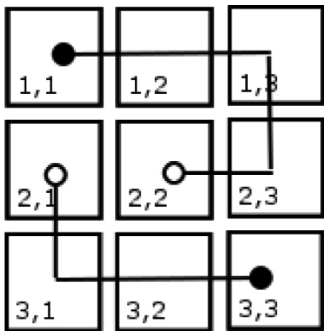

Fig. 2. A path coverage for the sample board — suboptimal one.

[1] There were no-one in the team who would cover all those roles alone.

[(2, 2), (2, 3), (1, 3), (1, 2), (1, 1), (2, 1), (3, 1), (3, 2), (3, 3)]

The task of the algorithm was to minimalize the number of the used paths. It can be observed that it is not possible to encode all of the path coverages, but it is possible to encode minimal path coverages in this way. With this representation it was possible to use Permutation Crossover (PMX) and Ordered Crossover (ODX) [5] operators.

Another representation of the path coverage is a graph representation. Permutations were treated as a paths in graph, so it was possible to use Edge Recombination Crossover (EROX) operator [15].

In the experiments there have also been tested different mutation operators like displacement and insertion, which operated on the permutational representation of the path coverages.

Results

An experiment was run in order to determine the best setup for a genetic algorithm i.e. which crossover, mutation and selection operator should be chosen. It turned out that the selection of the crossover operator has the greatest influence on quality of the results. All crossover operators were tested in the same settings. The full board was chosen as a benchmark. The algorithm which found the solution in the smallest amount of time was chosen for future evaluation.

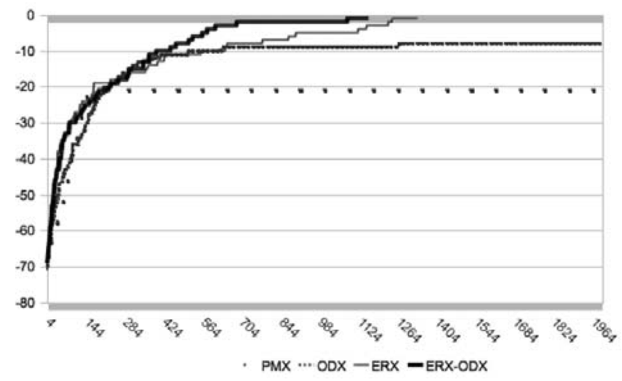

Fig. 3. Evolution of fitness function depending on choice of different crossover operators.

After the initial experiments it was observed that EROX operator caused algorithm to stall in some local minima, while ODX operator provided very chaotic exploration pattern. In brief, EROX showed exploatative properties, while ODX displayed more exploratative ones [2]. With those observations new combined EROX-ODX operator was created. It has chosen EROX or ODX operator depending on the variance of fitness in population. Lower variance may indicate that the algorithm is not improving and caused bigger probability of explorative crossover operator.

Further experiments, involving tests of new crossover operator, were carried out using same testing scenario. It turned out that combined EROX-ODX operator performs best of all in this task. Plot (3) shows evolution of fitness function through algorithm run.

When fitness function has a value of -1 it means that one path covering all board was found — which was optimal. Fitness function has the negative values, because it was inverted in order to have it monotonically growing.

Relying on these results ERX-ODX operator was chosen to include in final algorithm which was run to generate puzzles.

Disscussion

The new tool has enhanced level design process. The level designer was not forced to create new puzzles, but used the presented tool in order to generate different puzzles. After that, reviewer rated them and reviewed associated solutions. The level design process employing PCG was schematically shown in the Fig. 4. Created puzzles were also enhanced later by using hints and obstacles, as those elements of the board were not produced by PCG algorithm.

The new tool also impacted a testing procedure. There was a problem with testing, because an author of puzzle is usually not a reliable critic of own product. The previous approach required testing puzzles by other team members who played a role of puzzle critics. With the new approach the designer could play a role of the puzzle critic as he was not an author of a puzzle. This saved some work of 1 men, but in the team of 4 people this is much. Amount of testing iterations of single puzzle was reduced.

In presented case there was a risk of project failure an inherent trait of the research projects. Creation of a new tool for PCG should be aware decision of game the producer. In *Keri Tap* however, the project completed successfully and about 20% of the game content was generated with PCG — this ratio is not remarkable, but the decision of using PCG was made in halfway of the project development phase. There are plans to improve this method — the ultimate goal would be to generate puzzles with a predictable difficulty level on demand2.

Conclusions

The main contribution of this paper is to present PCG method as the tool, which is usable not only by large game producers with R&D departments, but also by

2 In a gameplay, not in a design phase.

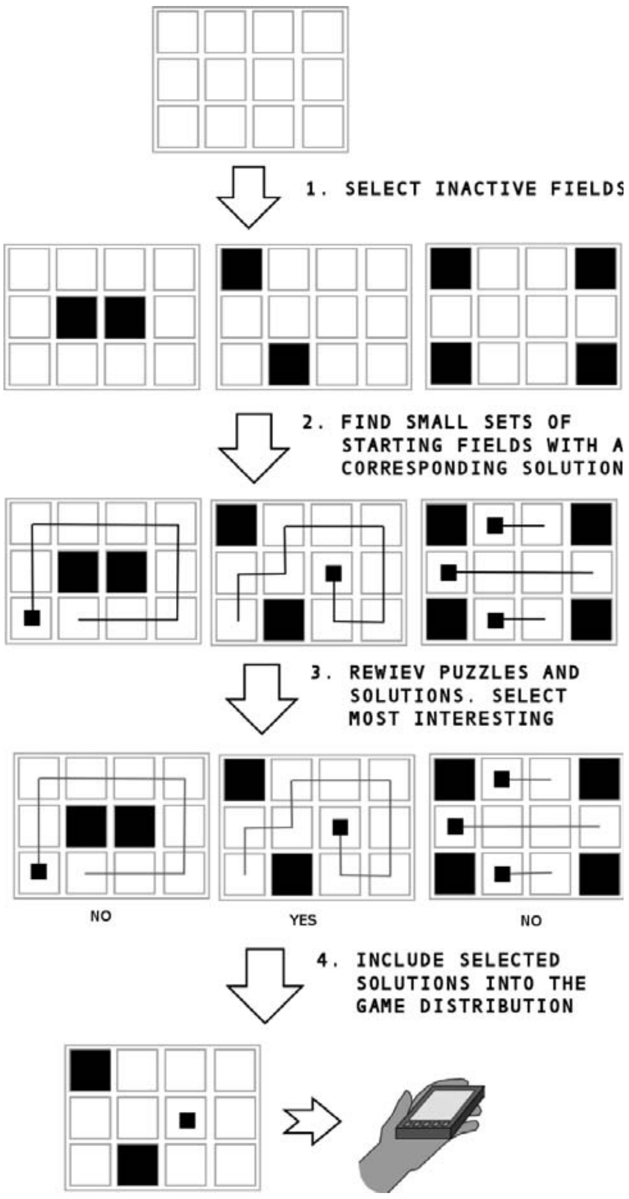
1. SELECT INACTIVE FIELDS
2. FIND SMALL SETS OF STARTING FIELDS WITH A CORRESPONDING SOLUTION
3. REWIEV PUZZLES AND SOLUTIONS. SELECT MOST INTERESTING
NO
YES
NO
4. INCLUDE SELECTED SOLUTIONS INTO THE GAME DISTRIBUTION

Fig. 4. Level design process with PCG method in Keri Tap.

small independent video game developers. This result is backed up with case study of *Keri Tap* game. In presented case-study level design process was simplified what resulted in time-saving, which could be spent on other issues.

The minor contribution was to show application of GA in very specific problem related to graph theory and also present application of this theory in game development industry. It's hard to generalize this solution for other games, but it's important to promote application of the mathematics to construct logical video games.

Acknowledgements

I would like to thank my mentor prof. Jarosław Arabas for helping me to do my research.

References

[1] Alon Itai, C.H.P., and J.L. Szwarcfiter. "Hamilton paths in grid graphs". *Society for Industrial and Applied Mathematics* 11 (4), 1982: 676–686.

[2] Arabas, J. "Predicting genetic diversity of populations in evolutionary search in R1 ". Proceedings of the 2011 Evolutionary Computation and Global Optimization conference, 2011.

[3] Arkin, E.M., et al. *Not being (super)thin or solid is hard: A study of grid hamiltonicity*, 2008.

[4] Ashlock, D., C. Lee, and C. McGuinness. "Search-based procedural generation of maze-like levels". *IEEE Trans. Comput. Intellig. and AI in Games* 3 (3), 2011: 260–273.

[5] Buckland, M., and M. Collins. *AI Techniques for Game Programming*. Premier Press, 2002.

[6] Goldberg, D.E. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[7] Gudlaugsson, B. *Procedural Content Generation*. Reykjavik University, 2006.

[8] Juul, J. "The open and the closed: Game of emergence and games of progression". In Proc. Computer Game and Digital Cultures, 2002: 323–329.

[9] Moraglio, A., J. Togelius, and S.M. Lucas. "Product geometric crossover for the Sudoku puzzle". Proceedings of the IEEE Congress on Evolutionary Computation (CEC), 2006.

[10] Nielsen, J. *Participation inequality: Encouraging more users to contribute*, 2006.

[11] Ondrejka, C., Ed. *Escaping the Gilded Cage: User Created Content and Building the Metaverse*. New York Law School, 2003.

[12] O'reilly, T. *What is web 2.0? design patterns and business models for the next generation of software*.

[13] Prusinkiewicz, P., and A. Lindenmayer. *The algorithmic beauty of plants*. New York, NY, USA: Springer-Verlag New York, Inc., 1996.

[14] Togelius, J., et al. "Search-based procedural content generation: A taxonomy and survey". *IEEE Trans. Comput. Intellig. and AI in Games* 3 (3), 2011: 172--186.

[15] Whitley, d., T. Starkweather, and D. Shaner "The traveling salesman and sequence scheduling: Quality solutions using genetic edge recombination". In: *In Handbook of Genetic Algorithms*, 1990: 350–372.

[16] Wirth, N. *Algorithms + Data Structures = Programs*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1978.