Procedural Level Generation for 2D Games

Csoka Ervin
Faculty of Computer Science – Babes Bolyai University
2020

Abstract

Games require dedicated teams of content creators and designers in order to provide a novel experience to the player. With increasing demand for content from consumers we turn to the procedural generation of game assets. The employed procedures should generate assets that satisfy the playability constraints required by a game as well as produce relatively novel outputs. We provide an overview on the usage of Occupancy-Regulated Extension, Novelty Search, Minimal Criteria Novelty Search and Feasible-Infeasible Two-Populations as generation techniques for game environment that is both valid and novel. Finally we present the results of Feasible-Infeasible Two-Populations used as a generation tool employed in a two-dimensional game engine.

Introduction

Level design typically requires the involvement of a team of dedicated designers. Constraints must be met regarding playabaility and, in addition, playing through a level must be considered a fun experience by the player. A level is built by arranging static components, that is 2D tiles in our case, and dynamic components such as enemies or other non-player-characters [2]. Tiles determine the way in which a level can be traversed. Their arrangement together with the placement of enemy characters determine the level playability, difficulty and player enjoyment, often requiring precision and speed. Thus constructing a level is both an optimization problem with respects to the amount of fun and difficulty it provides as well as a constraint satisfaction problem regarding it's playabaility.

As the demand for available game content rises, Procedural Content Generation (PCG) has become an option developers are increasingly taking into account in order to lessen the burden of designers or replace them altogether [7]. For the most part it has been used for the generation of game environment [4,8], though there are examples of PCG being used to generate game systems, game scenarios, visuals or items [9].

The algorithmic generation of assets provides benefits in terms of development cost and development time as well as cost of change, allowing designers to specify the properties a level should have, as opposed to how levels should be assembled. This can be achieved by merely tweaking some sets of parameters. What must be taken into account when employing PCG techniques is the difficulty of achieving the proper level of control over the generation of assets, the complexity involved in the development and testing of tools, the lack of standardized tools for game content generation as well as the fact that such methods tend to be domain-specific [1,2,7].

Our aim in this paper is to provide an overview of some of a number of tried content generation techniques as well as present the application of a generation tool in the context of a game engine focused on developing two-dimensional games.

Related Work

The use of PCG methods is noted in realizing systems such as particle systems, animation system and rendering systems in [1]. The animation system changes the animation of in-game characters according to a notoriety value associated to the player. Difficulties are encountered in the development of a procedural sky rendering pipeline and the assurance of correct algorithm outputs.

A tool for supporting the designer with suggestions in creating 2D maps through Novelty Search (NS) is proposed in [11]. It provides a map editor for the user to design the map. The tool evaluates playability constraints and gameplay properties, providing live alternative suggestions to the map design.

In [2] the Feasible-Infeasible Two-populations (FI-2Pop) genetic algorithm is employed with the goal of generating levels that meet constraints expressed in terms of difficulty and perceived fun, being general enough to be applied in a variety of games. It provides examples of generating entire levels by taking as input the specification of level elements as well as functions that define the desired constraints said elements must satisfy.

A geometry assembly algorithm applied in Infinite Mario is presented in [7]. Its main goal is to generate interesting geometry without knowledge of game-specific mechanics, focusing on variety over playability.

Games of the Rogulike genre are one of the most popular type of games that employ PCG. They can be considered a subcategory of the Role-Playing-Game (RPG) genre. The genre can be traced back to the 1980's game Rogue [4,9,11,12] which is a turn-based RPG with ASCII graphics that can generate a vast number of unique levels featuring enemies and collectibles. Discussions regarding what is and is not a Roguelike or what exactly are the core characteristics of such games do not have clear conclusions and an in-depth analysis of such is outside the scope of this work. However a list of Roguelike features has been assembled in 2008 at the International Roguelike Development Conference in Berlin, but the term is commonly used to describe games that have above average difficulty and provide procedural environment generation [12].

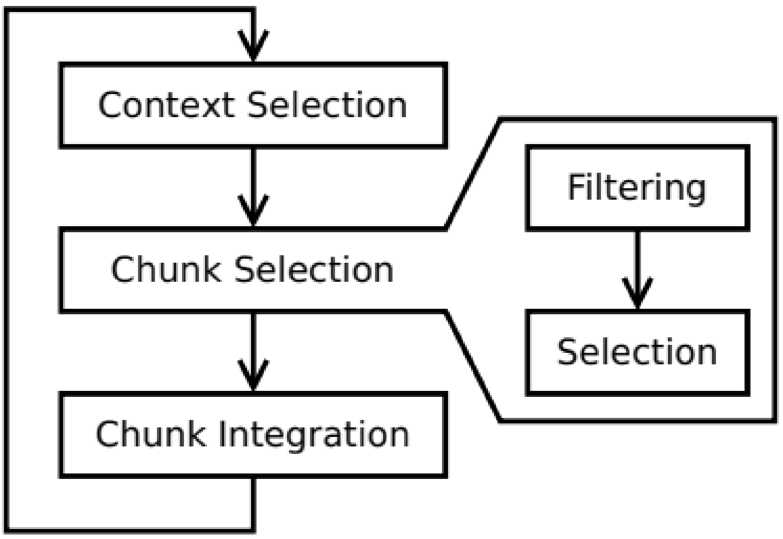
In what follows we provide an overview of the techniques we have taken into account that can be used to implement a content generation mechanism. We have searched for techniques that allow specifying constraints in order to achieve playability and have the potential to produce relatively diverse results without being bound to a single game. A desirable trait is that users should be able to influence the outputs, without requiring the aid of software engineers.

Generation Techniques

Occupancy-Regulated Extension

The main goal of Occupancy-Regulated Extension (ORE) is to generate diverse levels by assembling chunks of tiles provided in a library, where each chunk has associated anchors that denote potential locations the player might find himself in [7].

Chunks are selected from the library and placed in the level such that the anchors are preserved and content is iteratively generated. Chunk placement is achieved by selecting an anchor of the existing geometry to expand, searching through the library for one chunk that matches the existing content and finally integrating the chunk in the world. Chunk selection uses weights computed per chunk. They are obtained according to the annotation values in the library and to how many times a chunk was used. The chance that a chunk is selected is proportional to its weight.



Figure(1): ORE Chunk Selection Process. Source: [7]

The fact that ORE uses an authored library of elements can be considered both a strength and a drawback. On one hand it allows for custom complex components to be used and can also function as a partner for a designer, since the algorithm is not hindered by existing elements. As such, changes to level style require merely updating the library. Experiments [7] show interesting outputs, resulting in complex level patterns that introduce safe zones and layering [13]. On the other hand in order to use the algorithm a library of chunks must be created and maintained. In addition, ORE only strives to satisfy the constraints regarding anchor points and, depending on the mechanics available to the player, might generate environment that cannot be traversed.

Genetic Algorithms

A Genetic Algorithm (GA) appears as a good fit for implementing a content generation tool [3]. GAs mirror natural selection as they select the fittest individuals of each generation for reproduction. They are used to optimize an objective (fitness) function with multiple variables that represents the distance of an individual to some objective state. In the case of constrained optimization the usual approach is to penalize solutions proportionally to the graveness of violation.

The general form of a maximization constrained optimization problem is illustrated in Figure(2), where *Z(x)* is the fitness function value for individual *x* and *E,F,G* model constraints as inequalities that must be satisfied by a solution, with S_i representing the set of decision variables.

$$\max_{x_i} z = Z(\boldsymbol{x}), \text{subject to } E(\boldsymbol{x}) \geq \boldsymbol{a}, F(\boldsymbol{x}) \leq \boldsymbol{b}, G(\boldsymbol{x}) = \boldsymbol{c}, x_i \in S_i$$

Figure(2): The general form of a maximization constraint optimisation problem. Source: [3]

The penalty function in Figure (3) represents the problem in a way that allows for it to be directly encoded in the GA, where *W(x)* is the fitness of *x*, *P(x)* is the combined penalty of all constraint violations of *x* and *Z(x)*, as defined above, is the fitness of the individual. This can lead to quick rejection of solutions, which may be unwanted.

$$\max_{x_i} z = W(\boldsymbol{x}) = Z(\boldsymbol{x}) - P(\boldsymbol{x}),$$

Figure(3): Penalty function representation. Source: [3]

The genotype is represented linearly as an array of Design Elements (DE). A DE represents one element of a level, regardless of its size. This means that DEs can represent equally one block of a level as well as an entire room. A level is encoded by specifying the elements it should contain.

Crossover between the individuals of a population is used in order to obtain the population of a future generation of individuals. There are many ways of implementing a crossover mechanism of which we enumerate only a few. Single-point crossover divides the set of genes of two individuals into two sets and creates children that contain a division from each parent. K-Point crossover follows the same principle, but instead of creating two sets it results in K disjunct sets of genes which are passed to the offspring. Finally uniform crossover states that every gene is equally probable to be passed to the offspring.

Mutation operations are also used to modify the genes of an individual. The mutation of an individual typically has a small probability compared to crossover and results in some change to one of the genes of an individual.

Genetic algorithms are especially sensitive to how the fitness function is modeled and may produce undesirable results in the context of a vague definition of it [3]. It is argued in [5,6] that positive results can be obtained by abolishing such a function and instead searching only for novelty, that is, genetic diversity within a population.

Novelty Search & Minimal Criteria Novelty Search

Novelty search is an evolutionary approach used when fitness is difficult to quantify. Its main goal is that of exploring the search space as opposed to optimizing a fitness function [5] having as an objective to increase the diversity of the solution set by favoring individuals situated at a greater distance from the population. Solutions are generated such that a function of the average distance between *k* neighbors in a set of individuals is maximized, where *k* is chosen experimentally. The average distance *p(x)* for individual *x* is computed as in Figure (4), where μ_i represents the *i*-th closes neighbor of *x* and the *dist* function quantifies how different 2 individuals are.

$$\rho(x) = \frac{1}{k} \sum_{i=0}^{k} \operatorname{dist}(x, \mu_i),$$

Figure(4): The function of novelty. Source:[5]

Minimal Criteria Novelty Search (MCNS) [6] provides a more radical take on classic NS by stating that that if an individual does not satisfy the minimal criteria for reproduction then it should not be allowed to further reproduce, thus setting the fitness of individuals that do not qualify as solutions to 0. This can improve fitness values obtained through Unconstrained NS, but limits the effectiveness of the technique in high-constraint problems.

Both NS and MCNS seem reasonable options for implementing a generation system. Results [5,6,10,11] show that Unconstrained NS performs well in relatively small spaces. Still it suffers from a small set set of feasible individuals in larger spaces, as the likelihood of generating infeasible offspring from feasible parents is high due to the focus on exploring the search space. MCNS tends to perform random search on large maps as it cannot find feasible individuals. Nonetheless it yields a high number of feasible individuals due to mutations, though at the cost of offering low diversity.

Feasible-Infeasible Two-Population

FI-2Pop genetic algorithm [2,3,8,10] allows specifying a set of hard constraints and high-level goals that must be satisfied for the generation of a level that is considered playable. It maintains a population of 'fit' individuals which satisfy all constraints and a population of 'unfit' individuals which do not. The algorithm requires 2 fitness functions: one that drives change of the valid individuals towards a global optimum and one for targeting the satisfaction of the specified set of constraints for the invalid individuals. The individuals that were rejected due to a violation of constraints resulting from crossover or mutation are continuously evolved such that the number of violations is minimized. An individual that satisfies all the constraints is moved to the set of feasible individuals and continues evolution with the goal of optimizing itself. Diversity is maintained by frequent migration between populations and by the surviving genetic material in infeasible individuals.

The fitness functions are applied on the individuals based on the criteria specified by the level designer. The common practice is to evolve the infeasible individuals towards feasibility while the feasible one pursue novelty.

Application

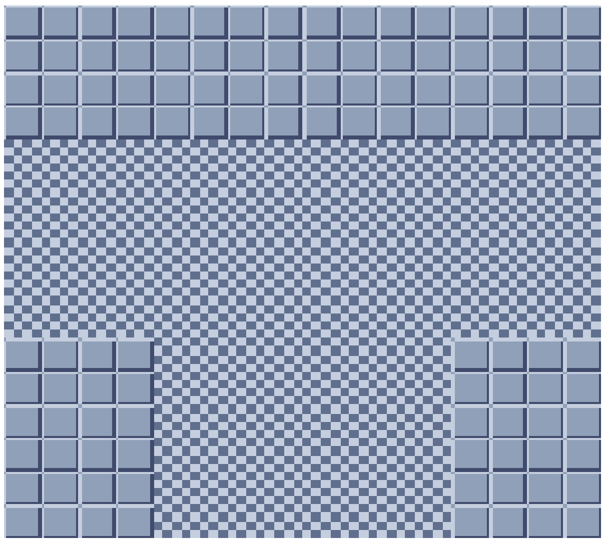
We have chosen to implement FI-2Pop as the generator for a two-dimensional game engine. As opposed to ORE, FI-2Pop does not require the management of a chunk library and only calls for specifying the tile types a map should have. FI-2Pop also avoids the random searches NS and MCNS tend to perform due to the genetic material maintained in the infeasible population. In addition, once a feasible individual is found the whole population starts moving towards it, providing better outcomes especially in small search spaces. This results in slightly lower diversity but more fit solutions.

In our case a DE in the genotype represents one of 11 types rooms, out of which 3 are considered safe zones for the player. Rooms contain different types of paths which might render them unreachable from other rooms.

$$p(x) = \frac{NV - CC + 1}{NV}$$

Figure(5): Fitness function that evaluates based on the strongly-connected components of a graph.

The constraints that a level must satisfy is that the player should be able to access any room from any other room. We modeled this by using a directed graph such that each room represents a node and every room type is annotated with the edges that are created when such a room is placed in the world. As such, in order to satisfy the above mentioned constraints, the rooms must be generated in such a way that the graph they create has only 1 strongly connected component. This is modeled as in Figure(5), where *NV* is the number of vertices of the graph, *CC* is the number of strongly connected components and *p(x)* is the fitness of individual *x*. Once a level satisfies all constraints it evolves towards distinguishing itself from the other individuals. In calculating diversity a level is rewarded for by 1 measure for having a different rotation compared to the other individuals and by 2 measures for using a different DE. Both rotation and element type can have a significant impact on how the level is traversed but we consider that seeing a different type of element produces a more



Figure(6): Design Element. Creates edges to the node below and the nodes to its left and right.

pronounced sense of novelty sensed by the player, as opposed to encountering the same element at a different angle.

We have obtained promising results for a tournament size equal to 4 with a total population of 25, crossover rate of 0.3 and mutation rate of 0.1, employing elitism to keep the two most promising individuals of a generation unchanged. The assumption is that these carry valuable genetic material and can be used to further increase fitness or diversity values. It must be noted that our tool is designed for use in a game engine and thus allows a user to modify these parameters to suit its needs. The results for our experiments are displayed in the table below, with feasible individuals being discovered in the first generation in every setup. The early discovery of feasible individuals could be a consequence of the way the rooms in this setup were designed, as nearly half the rooms generate paths to all their neighbors, making it relatively easy to achieve feasibility. We also noted that changing the population size did not affect this fact.

Changing the tournament size did not seem to have an effect on the results either. Eventually we decided to keep it at 4, assuming it would apply less selection pressure and thus give less fit individuals better reproduction chances. Another reason was efficiency, since even though the larger tournament did not affect results it did negatively affect the time needed to run the algorithm. We also noted that doubling the mutation rate from 0.1 to 0.2 did not significantly affect the results. Changing the room types so as to create tighter constraints might provide more interesting results in this respect as well.

On the other hand halving the mutation rate to 0.05 proved to significantly impact the time needed to assemble a complete feasible population while also lowering overall fitness. This suggests that individuals get stuck more often on local optima due to the lower mutation rate.

Tour. Size	8	4	4	4
Crossover Rate	0.50	0.50	0.40	0.50
Mutation Rate	0.10	0.10	0.10	0.05
Avg. Diversity	0.69	0.69	0.68	0.61
Avg. Fitness	0.70	0.70	0.69	0.67
Final Gen.	6	6	24	17

Finally reducing the crossover rate to 0.4 seemed to have the most impact on the ability of the algorithm to find solutions, requiring to run for 24 generations to fill the population and not providing any significant improvement to fitness or diversity.

These results confirm the fact that the algorithm can successfully be used to generate libraries of levels but also also hint to the fact that games could employ these techniques to generate levels on the spot.

Conclusion

In this paper we have presented five methods for generating game content, namely Occupancy-Regulated Extension, Genetic Algorithms, Novelty Search, Minimal Criteria Novelty Search and Feasible-Infeasible Two-Populations.

ORE uses annotated level chunks as the basis of its generation, focusing on the positions the player might occupy in relation to them, as such it is suitable to be used as a tool to support a human designer. Genetic Algorithms strive to optimize an objective function by simulating natural selection, but they are not suitable to high-constraint problems in their standard form. Novelty Search and Minimal Criteria Novelty Search eliminate the fitness function and choose to focus on novelty instead. They are methods suitable to be used when an objective function is hard to quantify and provide good results in relatively small search spaces.

FI-2Pop separates the search for novelty from the search for meeting constraints and allows feasible individuals to maximize their diversity while useful genetic material is kept and further evolved in the infeasible population. The results in our experiment show that results are found relatively fast and diversity is maintained even for longer periods of running the algorithm.

List of Figures

Figure(1): Chunk Selection Process. Source: [7]
Figure(2): The general form of a maximization constraint optimisation problem. Source: [3]

Figure(3): Penalty function representation. Soruce: [3]
Figure(4): The function of novelty. Source: [5]
Figure(5): Fitness function that evaluates based on the strongly-connected components of a graph.
Figure(6): Design Element. Creates edges to the node below and the nodes to its left and right.

Bibliography

[1] Chris Remo. "MIGS: Far Cry 2's Guay on the importance of procedural content" .In:Gamasutra(2008).

[2] Nathan Sorenson and Philippe Pasquier. "Towards a generic framework for au-tomated video game level creation". In:European conference on the applications ofevolutionary computation. Springer. 2010, pp. 131–140.

[3] Steven Orla Kimbrough, Ming Lu, and David Harlan Wood. "Exploring the evo-lutionary details of a feasible-infeasible two-population GA". In:InternationalConference on Parallel Problem Solving from Nature. Springer. 2004, pp. 292–301.

[4] Kate Compton and Michael Mateas. "Procedural Level Design for Platform Games."In:AIIDE. 2006, pp. 109–111.

[5] Joel Lehman and Kenneth O Stanley. "Efficiently evolving programs throughthe search for novelty". In:Proceedings of the 12th annual conference on Genetic andevolutionary computation. 2010, pp. 837–844

[6] Joel Lehman and Kenneth O Stanley. "Revising the evolutionary computationabstraction: minimal criteria novelty search". In:Proceedings of the 12th annualconference on Genetic and evolutionary computation. 2010, pp. 103–110.

[7] Peter Mawhorter and Michael Mateas. "Procedural level generation using occupancy-regulated extension". In:Proceedings of the 2010 IEEE Conference on ComputationalIntelligence and Games. IEEE. 2010, pp. 351–358 .

[8] Antonios Liapis, Georgios N Yannakakis, and Julian Togelius. "Neuroevolution-ary constrained optimization for content creation". In:2011 IEEE Conference onComputational Intelligence and Games (CIG'11). IEEE. 2011, pp. 71–78.

[9] Mark Hendrikx et al. "Procedural content generation for games: A survey". In:ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)9.1 (2013), pp. 1–22.

[10] Antonios Liapis, Georgios N Yannakakis, and Julian Togelius. "Enhancements to constrained novelty search: Two-population novelty search for generating gamecontent". In:Proceedings of the 15th annual conference on Genetic and evolutionarycomputation. 2013, pp. 343–350.

[11] Antonios Liapis, Georgios N Yannakakis, and Julian Togelius. "Sentient sketchbook: computer-assisted game level authoring". In:8th International Conferenceon the Foundations of Digital Games, Chania(2013).

[12] Xavier Ho, Martin Tomitsch, and Tomasz Bednarz. "Finding design influencewithin roguelike games". In:International Academic Conference on Meaningful Play.2016, pp. 1–27.

[13] Ahmed Khalifa, Fernando de Mesentier Silva, and Julian Togelius. “Level de-sign patterns in 2D games”. In:2019 IEEE Conference on Games (CoG). IEEE. 2019,pp. 1–8