# Grammar-based Procedural Content Generation from Designer-provided Difficulty Curves

Mircea Traichioiu
University of Amsterdam, Intelligent Systems Laboratory, Amsterdam, The Netherlands
mircea.traichioiu@gmail.com

Sander Bakkes
University of Amsterdam, Intelligent Systems Laboratory, Amsterdam, The Netherlands
s.c.j.bakkes@uva.nl

Diederik Roijers
University of Amsterdam, Intelligent Systems Laboratory, Amsterdam, The Netherlands
d.m.roijers@uva.nl

## ABSTRACT

In experience-driven procedural content generation (EDPCG) [15], the challenge of (parts of) a level are often subject to player-adaptive optimisation. However, this may interfere with the design goals of the game designer with respect to the difficulty build-up of the entire level, e.g., the designer may have specific ideas about where the climax of a level ought to be. This can be a reason for designers to not adopt experience-driven procedural techniques. In this paper we mitigate this, by meeting the designers half-way: the designers provide a set of allowed difficulty curves for a level, and decide where the AI is allowed to switch between these, e.g., after each or certain segments or only between levels. This way, the designer is in control of the tension levels and 'feel' of the level, while still allowing player adaptivity. This paper describes how to generate level(s) (segments) using difficulty curves, and how this can be applied to experience-driven procedural content generation. Experiments that validate our approach in an actual, open-source action-adventure game, reveal that it is consistently able to generate entire game levels that closely approximate distinct difficulty curves. Also, the adopted *generative grammar* approach ensures that the generated content will never be unplayable, as it results strictly from (presumably adequate) designer-provided grammars. Finally, the obtained experimental results show that the procedural generation of game levels consistently takes place in a reasonably computationally efficient manner. Given these obtained results, we conclude that our enhanced procedural approach provides an effective basis for generating game levels according to designer specifications, yielding new options for PCG.

## 1. INTRODUCTION

In this paper we present an approach to procedural content generation (PCG) [12] from designer-provided difficulty curves. The aim of the approach is to allow the game designer to guide the PCG process by providing a set of allowed difficulty curves for a level, and deciding where the AI is allowed to switch between these, e.g., after each or certain segments or only between levels. This way, the designer is in control of the tension levels and 'feel' of the level, while still allowing player adaptivity.

Our approach can be considered an instantiation of experience-driven procedural content generation (EDPCG) [15] under a mixed-initiative design perspective [13, 10, 14]. In this paper, we assume that game designers are able to provide a set of target *difficulty curves* of game levels, and a method for matching the right difficulty curves to the right player. As such, we employ a predefined difficulty curve – which is used in the generative process at design time – that can be updated while the game is being played in order to (re)generate the remaining parts of the level. The approach is based on generative grammars; they originate in linguistics, where they are used as a model to describe sets of linguistic phrases [5].

## 2. APPROACH

In the present work, the game world is specified through two distinct structures, the mission and the space (*cf.* Dormans and Bakkes [8, 1]). The mission encodes the types and order of the tasks which must be performed by the player in order to advance though the level. Based on a generated mission representation, the space describing the physical environment of the level is constructed. For both tasks, generative grammars are employed.

The main contribution of our proposed enhancement is the introduction of a target difficulty curve for guiding the generation process, allowing for a more precise control over the anticipated player experience. We employ a simple evolutionary method [9] (without cross-over) which mutates the mission components of the level. A population of level candidates is generated and mutated iteratively for obtaining improved solutions, i.e. solutions that closely follow the target difficulty curve.

The general structure of the approach is as follows. The game designer provides (1) the (initial) target difficulty curve of the level, (2) the general mission specification, (3) the mission rule set, and (4) the space rule set. The four design specifications are input into an evolutionary process that outputs a final mission graph through the use of generative grammars. The final mission graph is translated into the actual space of the level, in which the mission can be carried

out by the player.

**Mission specification**. The mission associated with a level describes the tasks which must be performed by the player in order to advance through the level. In an action-adventure game, these tasks usually consist of overcoming a group of enemies or solving a puzzle. Given this nature of the tasks, a difficulty measure can be associated to them by the game designer, reflecting e.g., the toughness of the enemies or the intricacy of the puzzle. We encode such a mission specification using a graph representation, in which vertices describe tasks, while edges denote the order in which the tasks will be encountered by the player.

To incorporate the difficulty-level associated with the tasks, as well as to allow a meaningful relationship between the mission and the space representation of the level, each node has a symbol associated with it. These symbols are defined by the game designer and represent the vocabulary onto which the employed generative grammar operates. The symbols can indicate either a terminal node or a non-terminal node. The terminal symbols denote level sections for which the space is ready to be generated and have a difficulty level associated with them. Non-terminal symbols need to be further expanded using generative mission rules, until the mission graph will contain only terminal nodes.

The rules operating on the mission graph form a Probabilistic Context Free Grammar [4]. As such, each rule has associated with it a left-hand-side non-terminal symbol and a right-hand-side subgraph consisting of terminal and/or non-terminal nodes. In order to ensure a proper transition from the non-terminal nodes to their associated subgraphs, the number of outbound edges of the left-hand-side non-terminal must be equal to the number of outbound edges of the right-hand-side subgraph. Additionally, each rule has an execution probability associated with it; it is a statistically neutral probability, unless the game designer decides to include a bias favouring specific rules. The sum of probabilities for all the rules having the same left-hand-side symbol adds up to 1.

**Difficulty curve specification**. A designer-flexible way to define a desired player experience can be achieved by constructing a target difficulty curve. Here, the curve encodes the anticipated difficulty encountered by the player as she advances through the level. Thus, the difficulty curve can be viewed as a function over time with values in a scalar difficulty range defined by the game designer.

The computation of the expected difficulty curve of a generated mission, under the previously described mission representation, is done by simulating a possible level traversal by the player. While the actual level exploration varies from player to player according to distinct player profiles, a reasonable approximation may be described by a Depth-First-Search (DFS) exploration of the mission graph. As such, each step of the DFS algorithm increments a time counter, where the difficulty associated with the current time reflects the difficulty of the currently encountered node. Given the discrete nature of time in this approach, the difficulty values of intermediate time points are obtained through linear interpolation. Considering the varying length of the possible generated levels, both the target and computed difficulty curves are normalized to a fixed time interval.

ALGORITHM 1: Mission generation with difficulty curves.

```
1  population = InitPopulation(mission_specification);
2  sort(population, fitness);
3  prevBest = -1;
4  epochs = 0;
5  epochsSinceImprovment = 0;
6  while epochs < maxEpochs && epochsSinceImprovement <
   maxImprovementEpochs && fitness(population[0]) >
   fitnessThreshold do
7      for i = 1 : size(population) * mutationPercentage do
8          idx = rand() modulo size(population);
9          mutate(population[idx]);
10     end
11     for i = size(population) : size(population) − (size(population)
       * discardPercentage) do
12         population[i] = new Individual(mission_specification);
13     end
14     sort(population, fitness);
15     epochs = epochs + 1;
16     if prevBest != fitness(population[0]) then
17         epochsSinceImprovement = 0;
18         prevBest = fitness(population[0]);
19     else
20         epochsSinceImprovement = epochsSinceImprovement + 1
21     end
22 end
```

**Mission generation**. In our approach, the actual generation of levels with respect to a target difficulty curve, and given a provided mission specification, is performed with an evolutionary method (Algorithm 1).

In the algorithm, an initial population is generated by applying the generative mission rules on the provided mission specification (containing non-terminal symbols), and by subsequently expanding all non-terminal nodes successively until only terminal nodes are left (as in Dormans and Bakkes [8, 1]). The population is subsequently improved through an iterative process. At each iteration a number of individuals is mutated and a fraction of the worst individuals (according to their fitness, explained later) is replaced by newly generated ones. To preserve genome expressiveness, no crossover is performed. This iterative procedure ends as soon as one of the following designer conditions is met: a maximum number of iterations has been performed, a good enough candidate (according to its fitness) was generated, or there has been no improvement in the fitness of the best individual for a given number of steps.

The mutation operation consists of selecting one of the subgraphs generated by a non-terminal node of the initial mission specification and regenerating it by applying a different sequence of rules (selected randomly according to their respective probabilities) to the corresponding non-terminal. For each individual, the node chosen for regeneration is the one responsible for the section with the greatest cumulative error between the resulting difficulty curve and the target difficulty curves. Note that since this operation can generate a worse individual than the previous one (due to the random nature of the application of rules), and since the iterative process can be stopped after a number of iterations, an 'any-time' solution is desirable. Thus, if the best individual in the population (according to its fitness) is chosen for mutation, a copy is created and then mutated, replacing

the worst individual (to avoid an increase in the population count). This way, the best individual obtained so far is not lost. Should the mutated copy proves to be better, then it will become the best individual of the population (i.e., elitism [7]).

The fitness of an individual is defined as the Root-Mean-Squared (RMS) deviation between the target difficulty curve and the difficulty curve of the individual. Thus, a lower value indicates a better individual. Note that a perfect fitness value usually cannot be attained during mission generation; it may not be possible to perfectly match a target difficulty curve, as this depends on how the mission specification and the mission rules are defined.

**Space generation**. Once evolutionary mission generation ends, the best candidate mission is determined and serves as a base for the generation of the level's space. For each terminal node in the mission graph a subspace is generated and then all subspaces are stitched together according to the spatial relationship between the nodes.

As each subspace is generated individually, distinct (procedural) methods may be employed for generating the actual game space per nodes, and the process may be parallelized. For our experiments, we will consider mission generation for top-down 2.5D action-adventure games. A more detailed description of this procedure is beyond the scope of the present paper; we kindly refer the reader to Dormans and Bakkes [8, 1] for an extensive description of the space generation process.

**Player adaptation**. The actual level generation process can be performed during the game's development process, but also online, i.e., while the game is being played. The latter ability provides a clear opportunity for personalising the procedurally generated mission to assessments on, e.g., the player experience. Upon input on assessments of the player's experience, domain knowledge of the game designer may define that the adopted difficulty curve should be shifted upwards, downwards, or should be replaced by an alternative curve altogether; all with the aim of providing a more balanced game experience while the game is being played.

Our approach enables online adaptation, with the limitation that during play of the game, the game's mission cannot be regenerated from the start, as the user has already played parts of the mission. The mission can, however, be straightforwardly segmented, with the next mission segment being on-the-fly generated dependent on player assessments and the resulting adjustments to the difficulty curve. In gaming practice, such segmentation of the procedural process can be achieved by adopting a game setup consisting of sequentially executed levels, or implementing (virtual) checkpoints.

## 3. EXPERIMENTAL SETUP

We implemented the procedural method developed by Dormans and Bakkes [8, 1] - and our subsequent enhancement - in the open-source game engine Free Libre Action Role-playing Engine (flare). The experiments concerned a linear mission specification, and a flexible set of mission rules (i.e., the mission rules potentially allow for highly distinct missions to be generated). The linear level consisted of a start and end node, connected by a chain of 10 identical non-terminal nodes. The set of mission rules involved the use of a single non-terminal symbol and 100 terminal symbols with associated difficulties ranging from 1 to 100. For each of these terminal symbols, two rules were generated, one producing the terminal node and one producing the terminal node connected to another non-terminal node. All of the production rules have the same probabilities of being chosen for expansion, although these probabilities can be adjusted by the game designer to favour longer or shorter generated levels. The mission specification and the aforementioned rule set are illustrated in Figure 1. The employed population size is 200, the mutation rate is 0.9, the discard rate is 0.1, and the maximum iteration count of 1000.

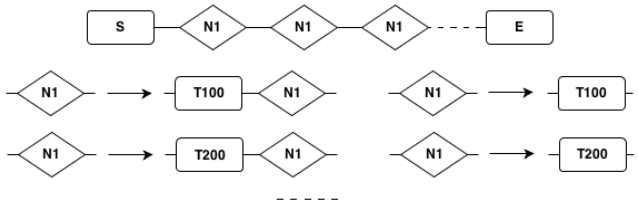


Figure 1: Mission specification and mission rules employed in the experiments. The ruleset involves the use of a non-terminal symbol (N1) and 100 terminal symbols, used in 200 corresponding production rules, from which only the first two pairs are depicted, using terminal nodes T100 and T200.

Four relatively complex target difficulty curves were experimented with: one based on a Gaussian distribution, one based on a sigmoid function, and two linear combinations, one involving an inverse and a sinusoidal function and another involving two sinusoidal functions. While we acknowledge that the selection of these difficulty curves was somewhat arbitrary, we presume that the expressiveness of difficulty over time as provided by the curves, is such that the experiments may indeed provide insight as to the performance of the proposed method in real-life game design scenarios.

The general form of the utilised fitness function is given in Equation 1. Two distinct variations of the function were tested. The first variation is based on the Root-Mean-Squared (RMS) difference between the candidate curve and the target curve (Equation 2). We refer to this first function as the RMS-based fitness function. The second variation implements an additional penalty in the score measure that is based on matching the sign of the curve's gradient (Equation 3). We refer to this second function as the smoothing fitness function.

$$\mathrm{fitness} = \sqrt{\frac{\sum_{i=1}^{N} \mathrm{score}_i}{\sum_{i=1}^{N} t_i^2}} \tag{1}$$

$$\mathrm{score}_i = (t_i - c_i)^2, i = 1..N \tag{2}$$

$$\mathrm{score}_i = (t_i - c_i)^2 \cdot \Big(1 + \Big|\frac{\mathrm{sgn}(t_{i-1} - t_i) - \mathrm{sgn}(c_{i-1} - c_i)}{2}\Big|\Big), i = 1..N \tag{3}$$

Table 1: Absolute error of generated solutions

| Shape of Difficulty Curve | RMS-based func. | Smoothing func. |
|---|---|---|
| Gaussian | 0.058686 | 0.072151 |
| Sigmoid | 0.043948 | 0.038735 |
| Inverse - Sinusoidal | 0.082256 | 0.097005 |
| Sinusoidal - Sinusoidal | 0.060651 | 0.053975 |

In the equations above, N represents the (normalized) length of the curve, $t_i$ denotes the value of the target curve at time i and $c_i$ denotes the value of the candidate curve at time i. This variation favours individuals whose difficulty curve not only matches the desired difficulty, but also the general shape of the target curve. This behaviour may be considered more desirable with respect to the player experience.

# 4. RESULTS

The results of adopting the RMS-based fitness function are given in Figures 2 (other figures omitted due to space limitations). We observe that in all cases, the curve of the generated mission closely follows the desired target curve. This is also expressed in terms of the absolute error of the final, generated solution, as given in Table 1. The results of adopting the smoothing fitness function are given in Figures 3 (other figures omitted due to space limitations). We observe that also in this setup, the curve of the generated mission closely follows the desired target curve. As expected, the smoothing fitness function provides an approximation with a more smooth general profile. As indicated by Table 1, the obtained absolute error when employing the smoothing fitness function does not differ substantially from the error obtained with the RMS-based fitness function. Given that the performance obtained with the smoothing fitness function and the RMS-based fitness function are comparable in terms of absolute error, one may surmise that game developers may opt for such a smoothing function, as it provides a less coarse profile in the actual difficulty curve. We observe that in every scenario, the generative approach appears to have converged to the best fitting solution within 1000 iterations. On standard computing hardware, i.e., an Intel Core i7-2675QM @ 2.2GHz, one iteration takes on average 28 milliseconds. This implies that on average our algorithm requires no more than 28 seconds for (re)generating a new mission. In several scenarios, the generative approach converged substantially faster; on average after approximately 575 iterations (i.e., approximately 16 seconds). Here, we note that the reported efficiency concerns the generation of an *entire* playable level. While it is undesirable to let game players wait for a next level for more than a few seconds, indeed, the generation process may be parallelised with actual gameplay, and can be constrained to (re)generate specific nodes or short segments. Under these caveats, we consider the developed method reasonably computationally efficient and lightweight for implementation in practise.

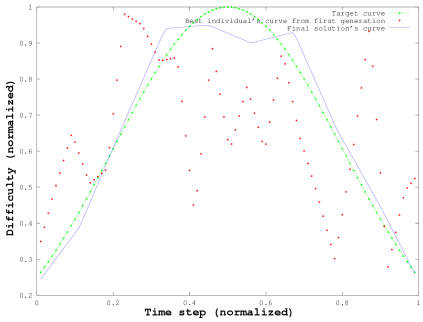


Figure 2: Obtained difficulty curves in the initial and final generation using a Gaussian-based target difficulty curve, with the RMS-based fitness function.

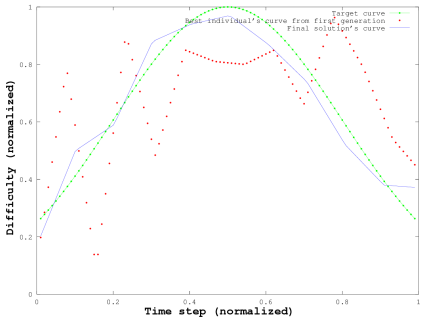


Figure 3: Obtained difficulty curves in the initial and final generation using a Gaussian-based target difficulty curve, with the smoothing fitness function.

# 5. CONCLUSIONS AND FUTURE WORK

We enhanced an existing approach for generating game levels with generative grammars. We defined three criteria for the enhanced approach, namely (1) it should be domain independent, (2) reasonably computationally efficient, and (3) procedurally generate content that strictly conforms to designer specifications with respect to a desired difficulty curve. Experiments that validated the enhanced approach in an actual, open-source action-adventure game, revealed that the approach is consistently able to generate entire game levels that closely approximate distinct difficulty curves. Also, the adopted generative grammar approach ensured that the generated content will never be unplayable, as it results strictly from (presumably adequate) designer-provided grammars. Finally, the obtained experimental results showed that the procedural generation of game levels consistently takes place in a reasonably computationally efficient manner. Given these obtained results, we concluded that our enhanced, domain-independent procedural approach provides an effective basis for generating game levels according to designer specifications.

For future work, we will investigate how multi-dimensional difficulty curves may be adopted, and how automatically generated player profiles may be incorporated in the grammar-based approach to procedural level generation. Taking into account additional characteristics of player behaviour such as non-deterministic action choice or past experience of the level would enable more expressive player models for evaluating perceived difficulty. Furthermore, we could reformulate the level generation problem with multiple objectives [6, 11], i.e., minimising the deviation from the target difficulty curves while maximising the appropriateness of challenge level to the individual player (either via implicit feedback [2] or emotion recognition [3]), thereby making the available trade-offs between the two explicit.

Further experimentation will also assess the performance in domains other than action-adventure games and the computational efficiency for more complex scenarios. Finally, finding a method for assisting designers in devising expressive enough grammars would also constitute a promising direction for future research.

## 6. REFERENCES


[1] S. C. J. Bakkes and J. Dormans. Involving player experience in dynamically generated missions and game spaces. In *Eleventh International Conference on Intelligent Games and Simulation (Game-On'2010)*, pages 72–79, 2010.

[2] S. C. J. Bakkes, S. A. Whiteson, G. Li, G. Visniuc, E. Charitos, N. Heijne, and A. Swellengrebel. Challenge balancing for personalised game spaces. In *Proceedings of the 6th IEEE Consumer Electronics Society Games, Entertainment, Media Conference (IEEE-GEM 2014)*, 2014.

[3] P. M. Blom, S. C. J. Bakkes, C. T. Tan, S. Whiteson, D. M. Roijers, R. Valenti, and T. Gevers. Towards personalised gaming via facial expression recognition. In *Proceedings of Tenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE'14), AAAI Press (Palo, 2014)*, 2014.

[4] E. Charniak. Statistical parsing with a context-free grammar and word statistics. *AAAI/IAAI*, 2005:598–603, 1997.

[5] N. Chomsky. Language and Mind (extended edition), 1972.

[6] C. C. Coello, G. B. Lamont, and D. A. Van Veldhuizen. *Evolutionary algorithms for solving multi-objective problems*. Springer Science & Business Media, 2007.

[7] K. Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001.

[8] J. Dormans and S. C. J. Bakkes. Generating missions and spaces for adaptable play experiences. *IEEE TCIAIG*, 3(3):216–228, 2011.

[9] E. D. Goodman. Introduction to genetic algorithms. In *Proceedings of the 2014 conference companion on Genetic and evolutionary computation companion*, pages 205–226. ACM, 2014.

[10] A. Liapis, G. Yannakakis, and J. Togelius. Sentient sketchbook: Computer-aided game level authoring. In *Proceedings of ACM Conference on Foundations of Digital Games*, 2013.

[11] D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, 2013.

[12] N. Shaker, J. Togelius, and M. J. Nelson. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2015.

[13] G. Smith, J. Whitehead, and M. Mateas. Tanagra: A mixed-initiative level design tool. In *FDG*, pages 209–216, 2010.

[14] G. N. Yannakakis, A. Liapis, and C. Alexopoulos. Mixed-initiative co-creativity. In *Proceedings of the ACM Conference on Foundations of Digital Games*, 2014.

[15] G. N. Yannakakis and J. Togelius. Experience-driven procedural content generation. *IEEE Tr. Aff. Comp.*, 2(3):147–161, 2011.