FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Procedural Generation of Tower Defense Maps with Measurable and Controlled Difficulty

**Carlos Miguel Sousa Vieira**

WORKING VERSION


Mestrado em Engenharia Informática e Computação

Supervisor: Prof. Alexandre Miguel Barbosa Valle de Carvalho

July 26, 2022

# Procedural Generation of Tower Defense Maps with Measurable and Controlled Difficulty

**Carlos Miguel Sousa Vieira**

Mestrado em Engenharia Informática e Computação

July 26, 2022

# Resumo

A indústria dos jogos cresceu consideravelmente na última década. Com a evolução da tecnologia de gráficos, complexidade e realismo, a procura por mais e melhor conteúdo tanto em tamanho quanto em detalhe também tem vindo a aumentar rapidamente.

Criar manualmente conteúdo para jogos é hoje em dia uma tarefa considerada lenta, cara, complexa e aplicada para conteúdo verdadeiramente original. Já requer atualmente muitos recursos humanos, como artistas, designers de som, designers de jogos, escritores de histórias e muito mais. Como alternativa, a utilização de técnicas de geração automática e complementar de conteúdos pela indústria dos jogos (por exemplo a geração procedimental) tem ajudado a atender a esta a procura por mais conteúdos. Estas técnicas permitem a geração de conteúdo através de algoritmos e procedimentos que necessitam de pouco input do ser humano.

Tower Defense (TD) é um subgénero dos jogos de estratégia que necessita bastante de conteúdo diferente e equilibrado no que diz respeito à progressão ao longo de mapas com graus de complexidade uniformemente crescente. O objectivo do deste tipo de jogos é eliminar enimigos que atravessam um percurso para chegar até à base do jogador através da colocação de torres no mapa. Sendo assim, a existência de mapas neste subgénero de jogos é crucial.

O presente trabalho pretende estudar técnicas para a geração automática de mapas para jogos de TD. Especificamente, o estudo inicia com a revisão do estado da arte em técnicas de geração de conteúdo para jogos relacionados. A partir desta revisão, as técnicas mais promissoras são estudadas e será desenvolvido um protótipo para testar e validar a geração de conteúdo procedimental de mapas que podem ter diferentes níveis de dificuldade. Os mapas serão gerados utilizando algoritmos de pesquisa e funções para avaliar o nível de dificuldade de um determinado mapa. Os mapas serão testados do ponto de vista de aferição de dificuldade por uma solução baseada em a Inteligência Artificial, que será também foco de atenção na revisão do estado da arte e perspetiva de solução.

Os resultados deste estudo contribuem para a criação automática de conteúdo para jogos digitais e subsequente validação automatizada do nível de dificuldade. Deste modo, diferentes mapas gerados podem ser oferecidos ao jogador numa ordem que apresenta um grau crescente e consistente de dificuldade.

# Abstract

The gaming industry has grown considerably over the past decade. With the evolution of technology in graphics, complexity, and realism, the demand for more and better content both in size and detail has also been increasing rapidly.

Creating content manually nowadays is considered a slow, expensive, and complex task and applied only to truly original content. It already requires a lot of human resources such as artists, sound designers, game designers, storytellers, and more. As an alternative, the usage of techniques of automatic content generation by the gaming industry (for example procedural generation) has helped meet this demand for more content. These techniques allow content generation through algorithms and procedures with little or indirect user input.

Tower defense (TD) is a sub-genre of strategy games that needs a lot of different and balanced content with respect to the progression through maps with ascending degrees of complexity. The goal of these kinds of games is to eliminate enemies that cross a path to reach the player's base by placing turrets strategically on the map. This way, the existence of maps in this sub-genre is crucial.

The present work aims to study techniques for the automatic generation of maps for TD games. Specifically, the study will start with reviewing the state of the art in content generation techniques for related games. From this review, the most promising techniques will be further studied and a prototype will be developed to test and validate the procedural content generation of maps that can have different levels of difficulty. Maps will be generated using searching algorithms and fitness functions to evaluate the difficulty level of a given map. The maps will be tested from the point of view of gauging difficulty by a solution based on Artificial Intelligence, which will also be the focus of attention in the review of the state of the art and solution perspective.

The outcomes of this study should contribute to the automatic content generation for digital games and subsequent automated difficulty level validation. This way, different generated maps can be presented to the player in an order with a consistent increasing difficulty level.

**Keywords:** Procedural Generation, PCG, Computer Games, Tower Defense, Map Generation

**ACM Classification:** CCS → Applied computing → Computers in other domains → Personal computers and PC applications → Computer games

*"Life is too short to play normals"*

Toamig

# Contents

# List of Figures

# List of Tables

# Listings

# Abreviaturas e Símbolos

| | |
|---|---|
| PCG | Procedural Content Generation |
| TD | Tower Defense |
| RTS | Real-Time Strategy |
| AI | Artificial Intelligence |
| RL | Reinforcement Learning |
| ML | Machine Learning |
| MCTS | Monte-Carlo Tree Search |

# Chapter 1

# Introduction

This work is focused on automatic Tower Defense (TD) map generation and automatic assessment of difficulty level. Towards this goal, section 1.1 presents a broader context, while section 1.2 describes the underlying problem that motivates the study. Section 1.3 details the hypothesis and questions to be answered while section 1.4 presents the perspective solution that addresses the problem described. Towards this purpose, section 1.5 presents the methodology to be carried out. Finally, section 1.6 describes the document structure of the following chapters.

## 1.1 Context

Video Games have been increasing in popularity over the past decades [4]. The game industry has evolved at a substantial rate, and it generated total revenue of $151.9 billion in 2019 [5].

Currently, games are used in different scenarios such as teaching or training specific skills, as it is the case of serious game applied to military, healthcare, education and more [6]. However, entertainment is a global concept that should remain present even in these kinds of games [7].

Overall, video games are a great form of entertainment for people of almost all ages due to the variety they can provide, among other aspects. Although it is not sometimes easy to classify games into strict families, there are a lot of different genres available for every taste [8]. The vast range of existing games in each genre, combined with each game's content, can ensure that each gaming session is almost unique. With this number of possibilities, gamers can spend hours playing different games or even the same one without getting bored.

Enjoyment and fun in games are relatively complex aspects to explain and measure. The pleasure retained from a single gaming session can vary from person to person. Although there is not a correct definition for these concepts [9] in video games accepted for everyone, there are some general considerations assumed by previous studies, for example, the flow theory [10, 11].

Considering the number of players, their different personalities, and tastes, it is possible to assume that the more content a game can have, the more likely it will be for a broader range of people to enjoy playing it. As for content, players do not only expect different environments, characters, weapons, or songs, but also content with more quality and a higher level of detail. For

the gaming industry granting fresh content is not the only concern. Balancing the content across all the different skill levels, from beginner to expert, is mandatory to avoid frustration or poor experiences overall [12].

## 1.2 Problem

In the game industry, constantly creating content has become a complex task with such a big and demanding target audience. The manual generation of content for games is mostly an expensive and slow process as it relies on the originality of game designers and creativity processes: a lot of factors influence what means good content and that depends on the game type, culture, artistic options, etc. Besides this, it may be also hard to assess the difficulty of the created game content (such as maps) and present these contents to players according to experience, skills, and desired higher levels of difficulty expressing challenge.

## 1.3 Hypothesis and research questions

Based on the context 1.1 and described problem 1.2 the study will concentrate on proving that, for TD games new content can be automatically created through PCG techniques, and for each newly generated map, it is possible to assess its associated difficulty level. While addressing the previously given hypothesis this study will also address the following research questions:

- What turns a map harder?
- How to generate a map from a certain difficulty level?
- How to assess and verify the difficulty of a generated map?

## 1.4 Solution Perspective

Procedural Content Generation (PCG) helps the game industry to keep up with the demand for new content by allowing automatic content generation through procedures and algorithms. These techniques aim to produce fast, reliable, controllable, expressive, and creative content by following a set of rules or controlled randomness [13]. The game designer can tweak these rules or parameters to control and improve the generated results.

Hence, based on the described problem this work aims to study PCG techniques towards automatic map generation for games from TD game genre. Previously, some ways of generating procedural TD maps and waves were presented using different approaches [14, 15]. Furthermore, the study will address how to assess the difficulty of a newly generated map. The goal is, for each map to measure the difficulty level and, for a set of maps, to present it in an ordered sequence of increasing difficulty.

## 1.5 Methodology

The solution perspective encompasses performing the following sequence of steps:

1. To research related work regarding the field of PCG towards automatic map generation. Also, to research related work regarding automatic assessment and evaluation of game difficulty.
2. Based on the related work, to design a solution for procedural TD map generation with the automatic assessment of difficulty level.
3. To design an experimentation based on a solution prototype (to be developed) that encompasses the selected techniques for automatic evaluation of generated maps and difficulty level estimation. Within this process, specify the relevant metrics to be retrieved during tests to be performed. Tests may be solely supported on running the prototype but may also require tests with users.
4. To conduct the experiment: perform tests and retrieve result data, analyze and discuss the results. To improve the solution and prototype and perform a new round of experiments, if needed.
5. During the stages previously mentioned, write the dissertation work document, including final conclusions, outcomes, and future work.

Regarding step 3, the solution prototype is estimated to go through the following development stages:

a) Develop/adapt a basic TD game engine, as the starting point of the prototype.

b) Implement the selected techniques for automatic generation of maps within the developed prototype.

c) Implement the selected techniques towards training a reinforcement learning model to play the developed prototype with the goal of measuring difficulty.

d) Assess difficulty of generated maps.

## 1.6 Document Structure

This document is structured in the following chapters:

- Chapter 1 introduces the problem with a context, the motivation for this work, the hypothesis, the solution perspective, and the methodology to be carried out during the process.
- Chapter 2 references previous related works and describes methods and techniques for PCG and automatic difficulty assessment for games in general and the TD sub-genre in specific.

- Chapter 3 describes the steps to be executed towards the development of the perspective solution and the work plan for this work.
- Chapter 4 concludes about this phase of the work.

# Chapter 2

# State of the Art

This chapter details the literature review performed through various preliminary studies that meet the goals of this work. To that end, section 2.2 describes approaches that can be related to the tasks of procedurally generating maps and paths for the game genre in the study (TD). Subsequently, section 2.3 makes a rundown through different methods of assessing the difficulty level in video games.

## 2.1 Video Games

Video games, also called "computer games" or "electronic games", have struggled to get a formal definition in the past. Although it might be easy to tell if something is or not a game just by looking at it, choosing the right words to define and encapsulate all existing games seems rather impossible. In previous studies, narratologists and ludologists came out with different theories about video games which always seemed to fail by the eyes of more recent investigators [16, 17].

Some of the definitions include:

> "... a mode of interaction between a player, a machine with an electronic visual display, and possibly other players, that is mediated by a meaningful fictional context, and sustained by an emotional attachment between the player and the outcomes of her actions within this fictional context." [18]

> "X is a videogame iff it is an artefact in a digital visual medium, is intended primarily as an object of entertainment, and is intended to provide such entertainment through the employment of one or both of the following modes of engagement: rule bound gameplay or interactive fiction." [16]

### 2.1.1 Genres

It's safe to say that the broad range of games and their different purposes and goals can sometimes generate a mismatch of expectations between everyone inside and outside the industry. There

are currently a lot of game genres and sub-genres of games that despite not being perfect, present a great help for everyone to perceive the overall plot of each game.

Gunn *et al.* present us with a taxonomy for games, considering genres and other related parameters to help the knowledge transfer between game developers and related areas, namely AI researchers [19]. They categorized the games into 9 game genres, being: Action, Adventure, Role-Playing, Vehicle-Simulation, Strategy, Management, 4X, Life-Simulation, and Puzzle.

### 2.1.2 Tower Defense

The TD sub-genre locates itself within the genre of strategy games, more specifically, Real-Time Strategy (RTS) games. Some examples of TD games are the iconic Bloons TD [20], Plants vs. Zombies [21], Kingdom Rush, and more.

The common goal shared by games within this sub-genre is to defend a base by preventing incoming waves of enemies from reaching or attacking it. In most games, some currency (money, gold, coins, iron) is obtained from killing enemies and can be used to upgrade the player's gear.

The most common version of TD games consists of a plain map with a path from a starting point (where enemies spawn) to the castle the player must defend. The enemies travel along this path and try to reach the castle while the player puts turrets near the path to slay the enemies. These turrets can be upgraded with currency, and the evolution usually provides more stats (range, damage, attack-speed) or effects that turn them more effective against the enemies.

Ideally, games from this sub-genre have no endgame, meaning they can be infinitely played. Each map should have a version with infinite procedural enemy rounds where the player can have fun beating gigantic waves of the most powerful enemies. If the maps are set to have a final round, there should always be new maps with different difficulties to clear.

### 2.1.3 Related work

In the past, some solutions of PCG were investigated targeting TD games. Among the surveyed studies:

- 4 generated and evolved enemy waves using search-based PCG [22, 23, 24, 15].
- 1 generated maps and paths using Perlin noise and A* [25].
- 1 generated maps using Reinforcement Learning (RL) [14].
- 1 generated maps using search-based PCG and evolutionary algorithms [23].
- 1 assessed generated map difficulty using Monte-Carlo Tree Search (MCTS) as an automated playtesting solution [15].

## 2.2 Map Generation

The field of procedural content generation (PCG) includes a variety of ways of generating terrains, maps or levels for games [26, 27]. This section focuses on exposing some of the most used techniques for this purpose, as well as their advantages and disadvantages. It also references previous works where these approaches were applied. The referenced PCG methods are organized according to the taxonomy elaborated by Shaker *et al.* [13].

### 2.2.1 Fractals and noise

According to Shaker *et al.* [1] fractals and noise techniques are good ways of generating terrains, landscapes, and maps in general. These methods are stochastic and fast but really effective for the intended purpose. Famous approaches like Perlin noise [28] and the diamond-square algorithm are part of this subsection of PCG.

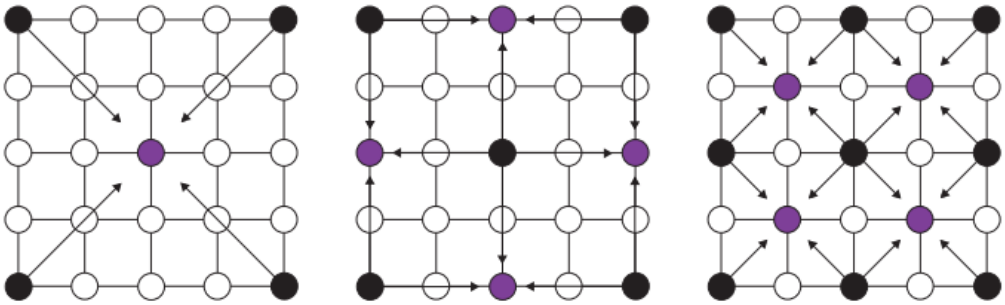
These methods are based on the usage of intensity maps to generate height maps. On the most basic approach, it is possible to generate a terrain based on a 2D array of randomly generated numbers from a desired range. Although this can be a really easy method and can actually provide a random terrain, this probably won't be of any use. Its high level of randomness will make it look completely unnatural and most likely not generate any familiar structures like hills or plains.

To overcome this unrealism and generate smoother terrains, it is possible to apply the same principles of the previous random terrain generation but this time with interpolation functions. The idea here is instead of generating every value for the height map, to subdivide it into regions and generate only its corner values. The remaining values should then be interpolated through any desired function according to the desired final result. The most frequently used interpolation methods are the bi-linear and bi-cubic interpolations. It is even possible to add an extra layer of smoothness by using a gradient-based interpolation. The principle of interpolation remains the same but this time instead of interpolating the heights directly, what is being interpolated is the rate of change in height.

The results obtained from the interpolation terrain generation are pretty good, especially within each region. However, when seen from a broader perspective, those terrains present an unnatural pattern due to the constant frequency of change. One way of solving this issue is the usage of fractal mathematics to generate similar features across different terrain scales. The diamond-square algorithm is one of the most known fractal techniques for terrain generation.

### 2.2.2 Constructive generation

Constructive generation encompasses methods of generating content that do not evaluate the obtained results. Due to this, they provide reduced control to the game designer [29] These methods tend to be fast and straightforward and are primarily used for dungeons and levels, which may contain a lot of randomness and few constraints.



(a) Diamond step (b) Square step (c) Diamond step repeats

Figure 2.1: The diamond square algorithm. (Illustration credit: Amy Hoover) [1]

The first family of algorithms within this category are the space partitioning ones. These algorithms consist in subdividing the 2D or 3D space into smaller regions recursively. *Binary space partitioning* (BSP) is the most popular space partitioning method, and it splits the area into two disjoint subsets allowing the space to be represented as a binary tree. These techniques are mainly used for generating non-overlapping content and categorizing zones due to the hierarchical tree representation.

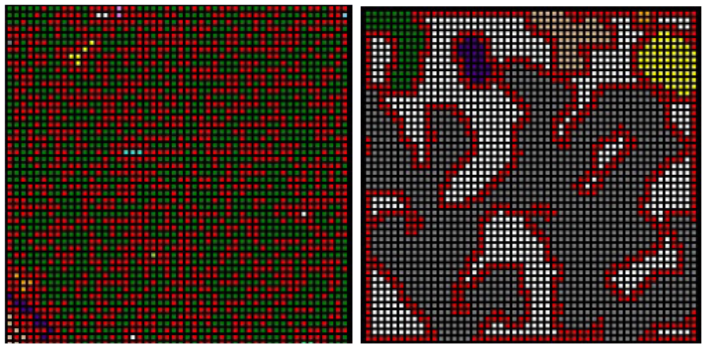
Another constructive generation approach relies on the usage of agents to dig paths and generate areas. This approach can create much more natural structures that tend to be chaotic depending on the adopted agent's behavior. It provides a great level of control, starting with the number of agents, their range of view, a bias to any direction, and much more:

> "There is an infinite number of AI behaviours for digger agents when creating dungeons, and they can result in vastly different results." [29]

Cellular automata are also excellent ways of generating structures with paths and obstacles like caves or dungeons but it has more usage. An automaton consists of a grid of n dimensions, some possible states, and some transition rules. It starts with an initial configuration and evolves its cells in each iteration based on the cell and its neighbors' current states according to the defined rules. This method can generate natural cave structures but comes with its problems. It's difficult to control and adapt the rules to produce the desired content.

> "... it is not easy to fully understand the impact that a single parameter has on the generation process, since each parameter affects multiple features of the generated maps. It is not possible to create a map that has specific requirements..." [29]

The last approach is the usage of generative grammars applied to procedural generation. This method describes the features to be generated in natural language. Like any other grammar, these model the content based on a set of recursive rules that define how bigger and smaller structures relate to each other.



(a) Random map (b) CA map

Figure 2.2: Cave generation: Comparison between a CA and a randomly generated map (r = 0.5 in both maps); CA parameters: n = 4, M = 1, T = 5. Rock and wall cells are represented by white and red colour respectively. Coloured areas represent different tunnels (floor clusters) [2].

### 2.2.3 Search-based approach

Search-based procedural content generation allows for the generation of controllable and evaluated content. It relies mainly on using evolutionary [30] and similar optimization algorithms to search for content with particular qualities [31].

This approach is made up of 3 important components:

- The *search algorithm* which finds and evolves the content until it fulfills the given requirements.
- The *content representation* which defines the representation of the content and features to be generated. It defines and limits the search space and what can be generated.
- The *evaluation functions* that indicate the quality of a given artifact.

The most used search algorithms for search-based problems are the evolutionary ones. These algorithms are based on Darwin's evolutionary theory and allow for an optimized search process. An initial population is generated at the start, and the best candidates are selected based on their fitness value. These elite candidates are them subject to genetic processes (*recombination*, *mutation* or *crossover*) and the method is repeated with the new population. If the search space is small enough, it might not be necessary to use genetic algorithms. In this situation, a simple "brute-force" algorithm that runs through every possible solution within the search space might be good enough.

The content representation is of great importance for the whole algorithm. It can completely change the efficiency and effectiveness of this approach. When defining the content representation for a given procedural generation system, the developer must consider a whole direct-indirect continuum. Although more direct representations may cover a broader range of more expressive solutions, the high dimensionality of large search spaces can make it harder to find reasonable solutions.

The evaluation functions are responsible for assigning a score (fitness value) to each possible solution. This given fitness value is based on the desired quality (e.g., difficulty) that the function should evaluate. To perform this evaluation, in search-based PCG, there are 3 different approaches:

- The *direct evaluation*.
- The *simulation-based evaluation*.
- The *interactive evaluation*.

Direct evaluation functions directly evaluate the content based on extracted features according to the quality to be assessed. These are the fastest to compute but hard to devise due to the complexity and subjectivity of gaming aspects. Within the direct evaluation, two types are to be considered: the *theory-driven* and *data-driven* functions, based on the source of information used to elaborate them.

Simulation-based evaluation functions rely on AI agents trained to excel on a specific task related to the content to be generated. These agents play the content that needs to be evaluated and provide statistics that allow its evaluation.

Interactive evaluation is similar to simulation-based but relies on human experiences instead of agents. The content is tested and evaluated by real people in this evaluation method.

### 2.2.4 Answer set programming

Answer set programming (ASP) is an approach to generating content based on constraints and logic. These constraints are usually written in *AnsProlog*, a language in which the syntax may resemble Prolog. A constraint solver then looks for all the possible configurations that meet the defined constraints. It is a great way of generating content within a smaller search space when the specific features for said content are known. [32]

### 2.2.5 Other

There are several other ways of generating procedural content for games, as are example the *L-systems* [33] and *Wave Function Collapse* [34]. These remaining methods don't quite match the goal of generating procedural TD maps, so they won't be covered in detail.

L-systems are a great approach to generating objects that appear a lot in games and despite looking identical, can have slightly different details. A great example that succeeded using l-systems is the *SpeedTree* middle-ware, which consists of a procedural system to generate and model vegetation.

Wave function collapse is an algorithm initially used for texture generation and is very well described by Sandhu *et al.*:

> "WavefunctionCollapse (WFC) is a non-backtracking, greedy search algorithm that is known for being able to use a small number of constraints to generate large consistent output" [35]

## 2.3 Automatic difficulty assessment

When creating different components for a game (e.g., maps, weapons, characters, enemies), it is essential to know how these components can affect the players or the game in general. For instance, a new weapon can have too much damage and become overpowered, and some maps can be impossible to complete, defeating the game's purpose.

A way to overcome all these design issues is by measuring the impact of newly generated content for the game. For this reason, automated playtesting and difficulty assessment play an important role alongside PCG when creating gaming content. There have been some studies in these fields, trying to measure the difficulty of different games and how to test a game in different ways [36, 37].

### 2.3.1 Difficulty in games

The concept of difficulty in games is rather abstract and can differ according to the game, the situation, or even the player itself. In their study, Constant *et al.* investigated the differences in player perception of difficulty [38]. When considering games that are not digital, like puzzles of any kind, most authors consider that the difficulty can be directly associated with the time taken to solve [39, 40, 41].

For digital games, with bigger environments and numerous possibilities, evaluating difficulty can be harder. Aponte *et al.* developed a model to evaluate the difficulty by subdividing the game into smaller challenges that should be easier to evaluate [9]. Roohi *et al.* confirm a correlation between difficulty and engagement [36], but none of them give a clear definition of difficulty for video games. Mourato *et al.* in his study defines difficulty in platform games, more scientifically, as being related to the probability of success in a given situation [42].

### 2.3.2 Difficulty evaluation

With the goal of studying difficulty estimation in puzzle games, Kreveld *et al.* [43] performed a literature review. According to their research, previous studies used automatic level solvers with metrics like the time taken or linear functions based on level features to evaluate each level's difficulty.

For their model of difficulty evaluation, they have chosen to define a function that takes the map or level as an input and gives the evaluated difficulty as an output. To better organize this process, they named three important steps:

1. Choosing the variables that will be used to estimate the difficulty in each game. These variables can be defined by having internal knowledge of how the game works either by playing it or by observing others. For reference, it is said that the ideal number of variables to make a good evaluation is between 4 to 7.

2. Choosing the appropriate model that correlates the chosen variables. For instance, Kreveld *et al.* point out two ways of combining these values: treating each variable independently

and weighing them or mixing multiple variables. In this study, they used a weighted linear combination of values as in 2.1.

3. Determining the weights for each variable by creating a model that, when given the variables, predicts the difficulty and compares it to the actual difficulty of each level. For their particular situation, the difficulty values were known from user studies.

$$Difficulty(L) = \sum(w_i \times V_i(L)) + w0 \tag{2.1}$$

### 2.3.3 Automated playtesting

In order to assess the actual difficulty of a given game, some way of playtesting is needed. This testing can either be performed traditionally, with humans, or automatically, recurring to machines and algorithms. Automated playtesting seems to solve the same problems as the automatic ways of content generation by avoiding the downsides of human work.

Albaghajati *et al.* [3] provided a framework that encapsulates all the previously known ways of automatically playtesting a game. The study categorizes all the found approaches in the literature and their purpose for the gaming industry.

#### 2.3.3.1 Approaches

*Search-Based* approaches can be utilized for game validation. These approaches are meant to explore the game in order to find a possible state that matches particular criteria. Evolutionary algorithms, graph-based algorithms, like Monte Carlo or A*, and other search algorithms are considered part of these approaches.

*Goal-Directed* approaches consist of more directed ways of searching. An agent is trained with specific goals through the definition of rewards and penalties, guiding its exploration. RL is probably one of the most famous methods to satisfy these criteria, although there are other algorithms that enforce a directed search process.

*Human-Like* approaches, just like its name, are approaches that focus on replicating behaviors that are exclusive from humans. These methods are used to mimic emotions like anger, curiosity, difficulty, and more. Machine Learning (ML) combined with human data can be used for this purpose.

*Scenario-Based* approaches are techniques that run specific tests based on or recorded from a human set of actions. In some situations, these approaches can work in a mixed-initiative scenario where a semi-automatic algorithm is running the game with the command of a human tester.

*Model-Based* approaches are used to verify the flow and logic of the game. For that, they transform the flow of the game into abstract formal models. These approaches usually use petri nets, game description languages (GDL), or unified modeling languages (UML).

| Approach | Implementations | Testing Objectives | Game Genres |
|---|---|---|---|
| **Search-Based** | • **Evolutionary Algorithms:** [16, 30, 31, 32, 39]<br>• **Graph search:**<br>  • Depth first: [41]<br>  • A*: [22, 35]<br>  • MCTS: [33, 34, 40]<br>  • Cicero: [42]<br>• **RRT:** [36, 37, 38] | • Functional correctness<br>• Game design correctness<br>• Game balance and fairness<br>• Progression and learnability | Sports, Turn-based, Cards, Arcade, Simulation, Tile-matching, MMORPG, Puzzle, Platform, Tower defense, Stealth, GVGAI |
| **Goal-Directed** | • **Reinforcement learning:** [47, 48, 49, 50]<br>• **Restricted heuristics:** [51, 52, 53] | • Functional correctness<br>• Game design correctness<br>• Game balance and fairness<br>• Progression and learnability<br>• Physics correctness<br>• Performance | Scene-based, Board, Tile-matching, Educational |
| **Human-Like** | • **Machine learning:** [54, 55, 20, 56, 57]<br>• **Restricted heuristics:** [58, 59, 60, 61, 62]<br>• **Mixed algorithms:**<br>  • MCTS and Stratabots: [63]<br>  • MCTS and Evolutionary Algorithms: [64]<br>  • MCTS and Machine Learning: [65] | • Functional correctness<br>• Game design correctness<br>• Game balance and fairness<br>• Progression and learnability | Sports, Shoot 'em up, Tile-matching, Sandbox, Puzzle, GVGAI, Cards, Adventure |
| **Scenario-Based** | • **Record and Replay:** [66, 67]<br>• **Game simulation:** [68, 69, 70]<br>• **Visual debugging:** [71, 72, 73] | • Functional correctness<br>• Game design correctness<br>• Visual correctness<br>• Multiplayer stability | Adventure, MMORPG, Sports, Arcade |
| **Model-Based** | • **Petri nets:** [74, 75]<br>• **Unified Modeling Language:** [76, 77]<br>• **Game Description Language:** [80]<br>• **Linear Temporal Logic:** [81]<br>• **ModelMMORPG:** [82, 83] | • Functional correctness<br>• Multiplayer stability | Educational, Platform, Scene-based, MMORPG, Puzzle |

TABLE I
SUMMARY OF THE STUDY FINDINGS

Figure 2.3: Summary of the study findings from Albaghajati *et al.* [3]

#### 2.3.3.2 Goals

According to Albaghajati *et al.* [3] categorization, developers can test games with the following objectives:

- Functional correction
- Multiplayer stability
- Performance
- Visual correctness
- Game design correctness
- Game balance and fairness
- Progression and learnability
- Physical correctness

#### 2.3.3.3 Summary

In his study, Albaghajati *et al.* [3] concluded that certain approaches are more suited to a determined testing objective than others. In particular, search-based seems to be the best approach for game balance and fairness in TD games, as it is possible to confirm in figure 2.3.

# Chapter 3

# Solution perspective and Work Plan

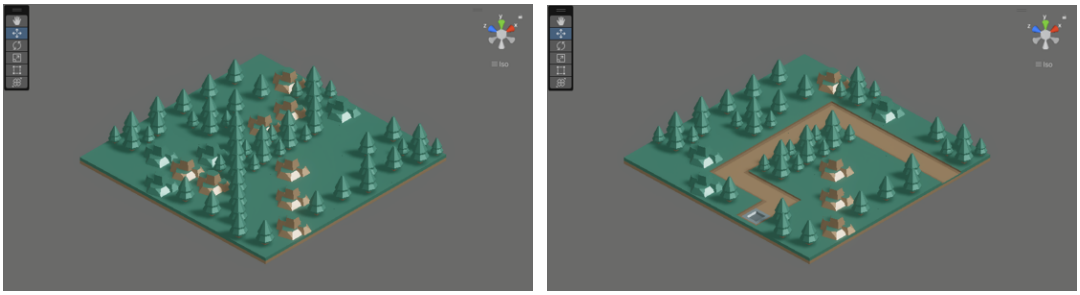
This chapter goes more in-depth into the implementation steps of the solution perspective and the work plan for the upcoming months.

## 3.1 Solution perspective

The perspective solution for the presented hypothesis 1.3 will require a number of steps for its execution. Those steps will encompass previously mentioned methods and techniques for map generation 2.2 and automatic difficulty assessment and estimation 2.3.

The expected workflow for the development of this solution will be:

1. Develop a basic TD game engine to serve as a test bench for the work.
2. Utilize procedural techniques from the fractals and noise approaches to generate map background terrains.
3. Utilize procedural techniques from the constructive generation approaches to generate paths.
4. Develop an automatic testing solution based on search-based methods to assess the difficulty level of generated maps.
5. Build a data-set of maps with known difficulty levels assessed by the previously developed tester.
6. Train a difficulty prediction model based on the built data-set that will associate features on the map with its difficulty.
7. Develop the full procedural system with search-based techniques that will output a map for the desired difficulty level utilizing the trained prediction model as the fitness function.



(a) Background terrain generation (b) Path generation

Figure 3.1: Experiments made on prototype map generation.

## 3.2 Experiments

Until the present moment, some preliminary experiments have been made regarding the terrain 3.1a and path 3.1b generation. These experiments prove that it is possible to generate simple maps recurring to basic stochastic and constructive PCG methods.

## 3.3 Work plan

The work plan for the upcoming months of this work is represented in figure 3.2.

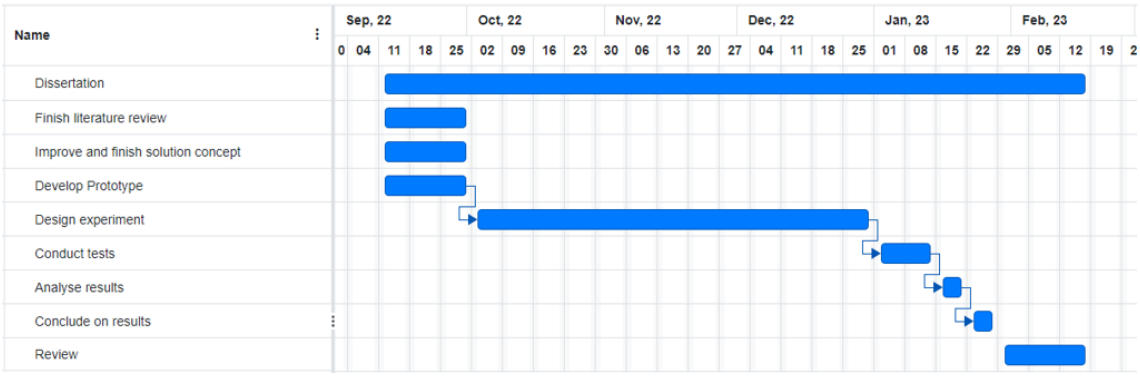



Figure 3.2: Work plan for the upcoming months.

# Chapter 4

# Conclusions

This phase showed the significance of this work for the procedural generation of gaming content with automatic difficulty level validation. Despite PCG being a great research area and the existence of studies on TD games, the pre-generation difficulty assessment of contents seems to be rather unexplored. This research showed that there were yet no public PCG solutions with a previous evaluation of difficulty for TD levels/maps. The literature review revealed a great connection between PCG and playtesting in games alongside difficulty estimation for balancing purposes. In order to develop a complete PCG system able to satisfy the entire player base, all these study areas should be combined into a complex solution. The deep exploration of all these approaches and techniques was great to have a more solid solution perspective and detail a better work plan.

# References


[1] Noor Shaker, Julian Togelius, and Mark J Nelson. Fractals, noise and agents with applications to landscapes. In *Procedural Content Generation in Games*, pages 57–72. Springer, 2016.

[2] Lawrence Johnson, Georgios N Yannakakis, and Julian Togelius. Cellular automata for real-time generation of infinite cave levels. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, pages 1–4, 2010.

[3] Aghyad Mohammad Albaghajati and Moataz Aly Kamaleldin Ahmed. Video game automated testing approaches: An assessment framework. *IEEE Transactions on Games*, 2020.

[4] Josh Howarth. 50+ amazing video game industry statistics (2022), march 2022. Available at https://explodingtopics.com/blog/video-game-stats, last accessed in july 2022.

[5] Deyan Georgiev. How many people play video games [video game statistics], january 2022. Available at https://review42.com/resources/video-game-statistics/, last accessed in May 2022.

[6] Tarja Susi, Mikael Johannesson, and Per Backlund. Serious games: An overview. *Skövde, Sweden: University of Skövde School of Humanities and Informatics*, 2007.

[7] Kristi Larson. Serious games and gamification in the corporate training environment: A literature review. *TechTrends*, 64(2):319–328, 2020.

[8] Rachel Ivy Clarke, Jin Ha Lee, and Neils Clark. Why video game genres fail: A classificatory analysis. *Games and Culture*, 12(5):445–465, 2017.

[9] Maria-Virginia Aponte, Guillaume Levieux, and Stephane Natkin. Measuring the level of difficulty in single player video games. *Entertainment Computing*, 2(4):205–213, 2011.

[10] Penelope Sweetser and Peta Wyeth. Gameflow: a model for evaluating player enjoyment in games. *Computers in Entertainment (CIE)*, 3(3):3–3, 2005.

[11] Mihaly Csikszentmihalyi and Mihaly Csikzentmihaly. *Flow: The psychology of optimal experience*, volume 1990. Harper & Row New York, 1990.

[12] Anthony Youssef and Stephen Cossell. Thoughts on adjusting perceived difficulty in games. In *Proceedings of the Sixth Australasian Conference on Interactive Entertainment*, pages 1–4, 2009.

[13] Noor Shaker, Julian Togelius, and Mark J Nelson. *Procedural content generation in games*. Springer, 2016.

[14] Xu Yueming and Tanaka Tetsuro. Procedural content generation for tower defense games: a preliminary experiment with reinforcement learning. *Game programming workshop 2021 Proceedings*, (2021):93–97, 2021.

[15] Simon Liu, Li Chaoran, Li Yue, Ma Heng, Hou Xiao, Shen Yiming, Wang Licong, Chen Ze, Guo Xianghao, Lu Hengtong, et al. Automatic generation of tower defense levels using pcg. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, pages 1–9, 2019.

[16] Grant Tavinor. Definition of videogames. *Contemporary Aesthetics (Journal Archive)*, 6(1):16, 2008.

[17] Jonne Arjoranta. How to define games and why we need to. *The Computer Games Journal*, 8(3):109–120, 2019.

[18] Rafaello Bergonse. Fifty years on, what exactly is a videogame? an essentialistic definitional approach. *The Computer Games Journal*, 6(4):239–255, 2017.

[19] EAA Gunn, BGW Craenen, and E Hart. A taxonomy of video games and ai. In *AI and Games Symposium*, pages 4–14. Citeseer, 2009.

[20] Bloons Tower Defense 6. [Android, iOS, macOS, Windows App], 2018.

[21] Plants vs. Zombies. [Android, iOS, macOS, Windows App], 2009.

[22] Yu Du, Jian Li, Xiao Hou, Hengtong Lu, Simon Cheng Liu, Xianghao Guo, Kehan Yang, and Qinting Tang. Automatic level generation for tower defense games. In *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pages 670–676. IEEE, 2019.

[23] Vid Kraner, Iztok Fister, and Lucija Brezočnik. Procedural content generation of custom tower defense game using genetic algorithms. In *International Conference "New Technologies, Development and Applications"*, pages 493–503. Springer, 2021.

[24] Supeno Mardi SN Ariyadi, Mauridhi H Purnomo, and SM Nugroho. Creep offenses evolution in tower defense games using nsga-ii. In *Proceedings of 8th International Conference on Information & Communication Technology and Systems (ICTS)*, pages 153–157, 2014.

[25] Michael Budiono, Liliana Liliana, and Hans Juwiantho. Meningkatkan kesulitan serangan enemy dengan menambahkan influence map pada metode a* pada procedural generated tower defense game. *Jurnal Infra*, 10(1):210–216, 2022.

[26] Sarah Katerji. *Procedural Content Generator For Platformer Levels*. PhD thesis, 2022.

[27] Izabella Antoniuk and Przemysław Rokita. Procedural generation of multilevel dungeons for application in computer games using schematic maps and l-system. In *Intelligent Methods and Big Data in Industrial Applications*, pages 261–275. Springer, 2019.

[28] Ken Perlin. Improving noise. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 681–682, 2002.

[29] Noor Shaker, Antonios Liapis, Julian Togelius, Ricardo Lopes, and Rafael Bidarra. Constructive generation methods for dungeons and levels. In *Procedural Content Generation in Games*, pages 31–55. Springer, 2016.

[30] SN Sivanandam and SN Deepa. Genetic algorithms. In *Introduction to genetic algorithms*, pages 15–37. Springer, 2008.

[31] Julian Togelius and Noor Shaker. The search-based approach. In *Procedural Content Generation in Games*, pages 17–30. Springer, 2016.

[32] Mark J Nelson and Adam M Smith. Asp with applications to mazes and levels. In *Procedural Content Generation in Games*, pages 143–157. Springer, 2016.

[33] Julian Togelius, Noor Shaker, and Joris Dormans. Grammars and l-systems with applications to vegetation and levels. In *Procedural Content Generation in Games*, pages 73–98. Springer, 2016.

[34] Maxim Gumin. Wave Function Collapse Algorithm, 9 2016.

[35] Arunpreet Sandhu, Zeyuan Chen, and Joshua McCoy. Enhancing wave function collapse with design-level constraints. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, pages 1–9, 2019.

[36] Shaghayegh Roohi, Christian Guckelsberger, Asko Relas, Henri Heiskanen, Jari Takatalo, and Perttu Hämäläinen. Predicting game difficulty and engagement using ai players. 5(CHI PLAY), 2021.

[37] Dagmara Dziedzic and Wojciech Włodarczyk. Approaches to measuring the difficulty of games in dynamic difficulty adjustment systems. *International Journal of Human–Computer Interaction*, 34(8):707–715, 2018.

[38] Thomas Constant, Guillaume Levieux, Axel Buendia, and Stéphane Natkin. From objective to subjective difficulty evaluation in video games. In Regina Bernhaupt, Girish Dalvi, Anirudha Joshi, Devanuj K. Balkrishan, Jacki O'Neill, and Marco Winckler, editors, *Human-Computer Interaction - INTERACT 2017*, pages 107–127, Cham, 2017. Springer International Publishing.

[39] Daniel Ashlock and Justin Schonfeld. Evolution for automatic assessment of the difficulty of sokoban boards. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2010.

[40] Timo Mantere and Janne Koljonen. Solving, rating and generating sudoku puzzles with ga. In *2007 IEEE congress on evolutionary computation*, pages 1382–1389. IEEE, 2007.

[41] Petr Jarušek and Radek Pelánek. Difficulty rating of sokoban puzzle. In *STAIRS 2010*, pages 140–150. IOS Press, 2010.

[42] Fausto Mourato and Manuel Próspero dos Santos. Measuring difficulty in platform videogames. In *4. ª Conferência Nacional Interacção humano-computador*, 2010.

[43] Marc van Kreveld, Maarten Löffler, and Paul Mutser. Automated puzzle difficulty estimation. In *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 415–422, 2015.