POLYTECHNIC UNIVERSITY OF MILAN

Master's Degree in Computer Science and Engineering

Master's Degree Thesis

Collaborative Maze Project: Computer-Aided Design system for a collaborative game.

Supervisors

Prof. Pier Luca LANZI

Prof. Daniele LOIACONO

Candidate

Umberto PICARIELLO

Matr, 883754

Academic year 2019/2020

"«In the end»?
***Nothing** ends, Adrian. Nothing **ever** ends."*

-Dr. Manhattan, Watchmen (1986) by Alan Moore

Summary

This thesis work represents the analysis and realization of a tool that supports designers in developing and testing a cooperative virtual reality experience. With the goal of simplifying the user's life, *Computer aided design* (CAD) employs the usage of a computer system (specifically a software) to aid the designer during its designing process, which is the act of creating, modifying, analyzing and optimizing a design. In this framework we have collaborated with a startup called *AnotheReality*: thanks to this collaboration, we have been able to realize a CAD software capable to assist game and level designers of the company in describing, testing and inspecting a virtual reality cooperative experience. This software is called *Collaborative Maze Project* and offers an easy and understandable User Interface on a tool that cooperates with designers. Rather than replacing them, all the components were developed with the scope of cooperating with the designers, that will always have the final word on the quality of the generated content. The designers can easily test different cases, game scenarios, and, through their validation, they will able to tune all the game generation parameters in order to get an high quality final product. Specifically, this work starts by giving a theoretical background to the final software realized, and it ends with a series of validation tests, with some space for future developments.

Acknowledgements

I want to give my thanks to Prof. Lanzi, to Dr. Loiacono and to Fabio Mosca of AnotheReality for their help and for having given me the opportunity to develop this work.

I also want to thank my family, since without them it would have been impossible for me to reach this goal, especially I want to thank my sister for revising the English version of this work.

Finally, I want to thank all the people I met during this master's, which have helped, supported and encouraged me during this journey. Especially I want to shout out to Andrea, Enrico and Ennio. A special thank goes to the people I had the fortune to share my first, small (but important to me) attempts to develop videogames with: the first one with Erick, Paolo and Alexander, and the second one with Angelo, Lorenzo T., Lorenzo V. and Luciano.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Videogames as a media is an increasing growing market, where the request for new, diverse and entertaining content is a constant. This request means a lot of work lying on the shoulders of designers, which feel the pressure to continuously produce a huge amount of high quality content. *Procedural Content Generation* (PCG) can be a good answer to this problem, but sometimes it cannot be enough by itself, since in some cases the best possible outcome can be reached only by designers. In all this cases a *Computer-Aided Design* (CAD) system can be a valid tool to fully exploit the computational power of a computer system without the concern of totally replacing humans, rather than supporting them. In this new framework, the computer assumes the role of effectively helping designers in co-creating content for them, while supporting their creative process. Our work has the goal to realize such a tool for a startup company: in particular, for a virtual reality cooperative game experience. Many CAD system have been developed for different fields, but the synergy with the PCG applied to videogames is pretty recent, and this work means to explore the potentiality of a CAD system applied to a particular videogame. That is why we have found fruitful the collaboration with a studio specialized in the development of immersive solutions, *AnotheReality*, since it allowed us to explore futher the usage of PCG in videogames and its synergy with CAD systems.

In developing this work, we will start by delineating a theoretical background to the use of PCG and CAD, which will later be applied to the videogame prototype proposed by *AnotheReality* and developed by the author of this thesis. After that, we will proceed to describe the structure and the main mechanics of the prototype, just after having given a brief introduction on the game genre of (*cooperative puzzle games*) in general. Later we will move on to outlining the CAD system implemented in its main core components, and we will conclude by listing some experiments done on the optimizer component of the CAD.

1.1 Outline

The thesis is organized as it follows:

In *chapter 2* we discuss the state of the art about puzzle games, collaborative puzzle games and procedural content generation.
In *chapter 3* we define the problem and explain the CAD framework with our intentions and purposes.
In *chapter 4* we completely show the tool realized: we describe the components and explain how they works, their roles in the CAD software and their final goal.
In *chapter 5* we have done some experiments to validate and support some choices made in the previous chapter.
In *chapter 6* we make our conclusions and describe future and possible development of the Collaborative Maze Project CAD system.

Chapter 2

State of the art

In this section we introduce briefly what puzzle games are and especially the cooperative feature that they host. After that, the Procedural Content Generation (abbreviated PCG) as a technique will be introduced, so as its use in general and in the videogame industry particularly. Subsequently, we will focus mainly on the thesis work, analyzing the *collaborative puzzle* games genre and how PCG can fit in this two worlds with the actual state of the art. Finally we will discuss map generation, with a particular focus on the thesis work framework.

2.1 Puzzle games and cooperative approach for team building

Technically speaking, *puzzles* are problems or challenges where the player has to find a solution given a previous knowledge or by properly explore the space of possible solution for the given problem[1]. As a matter of fact, when you think of "puzzles", your mind goes immediately to games. Jesse Schell in his book *The Art of Game Design A Book of Lenses* states that yes, puzzles require problem solving activities, but more precisely "A Puzzle Is a Game with a Dominant Strategy". "Both puzzles and games have problem solving at their core but puzzles are just miniature games whose goal is to find a dominant strategy": a set of actions (i.e. strategy) that, according to Schell, allows to beat the game every time, always resulting in a solution. Examples of mainstream puzzle games (both digital and

not) are: *Tetris*[1], *Puzzle Bobble*[2], *Rubik's Cube*[3], *Jigsaw*[4], *mazes*[5], *crossword*[6] and many others. Allowing two or more players to cooperate towards the same goal, defines a subgenre of puzzle games: cooperative puzzle games. We can use the previous definition of puzzles and consider the study promoted by Sedano et al.[2] to state that cooperative games are "the activity in which individuals come together to produce a single outcome, " and in the context of puzzle games that means people collectively trying to find the dominant strategy of the game. Examples of mainstream cooperative puzzle games are: emphPortal 2[7], emphTrine (videogame series)[8], emphKeep Talking and Nobody Explodes[9] and many others. An additional clarifying note must be done on the term "cooperative" used to define some sorts of puzzles, since the concepts of "cooperative games" and "dominant strategy" in the scientific literature most commonly refer to game theory, and some may confuse them with the definition of puzzles games we are referring to. Zaga et al.[3] distinguish two types of games in game theory: *non-cooperative/competitive* (every player wants to maximize its utility) and *cooperative* (players can form alliances, maximizing their respective utilities, but the game itself does not guarantee whether they will benefit or not from the cooperation). Accordingly to Zaga et al. a third category has been considered lately in game theory, called *collaborative games*: rewards and penalties are shared between the members that try to maximize the team's utility. It's clear that the difference between collaborative and cooperative games exists in game theory, but, outside of this context (that is if we consider the "game" as a form of entertainment) the terms are used interchangeably[2], and the meaning of "cooperative games" is closer to the one given a while ago for collaborative games. During this thesis we never consider this subtle distinction between cooperative and collaborative games, treating them as synonyms; and even for the definition "dominant strategy" given by Jesse Schell for puzzles, the reader will keep in mind that the concept is different from its use in game theory.

Finally, on the subject of clarifying definitions, when the word "game" will be used in this thesis, it must not be confused with the notion of "game" from the

[1]AcademySoft. "Tetris". AcademySoft, 1984.

[2]Taito. "Puzzle Bobble". Taito, 1994.

[3]E. Rubik. "Rubik's Cube". Rubik's Brand Ltd, 1980.

[4]Jigsaw puzzle.

[5]Maze.

[6]Crossword.

[7]Valve. "Portal 2". Valve, 2011.

[8]Frozenbyte. "Trine (videogame series)". Nobilis, Frozenbyte, Modus Games, 2009-2019.

[9]Steel Crate Games. "Keep Talking and Nobody Explodes". Steel Crate Games, 2015.

game theory framework, but rather as a form of play and entertainment.

2.1.1 Team building

While team training is used in many companies and organizations to improve team competences and processes, other team-affecting methods can be used in order to improve interpersonal relations and effectiveness of a work group[4]. Team building is an intervention focused on improving team functioning, mainly its operations and processes. Specifically, team building seeks to improve team processes, individual and team characteristics, and to change work structure or task[5]. Studies have underlined how team building activities focus on one (or more) of six components to achieve the "promised" improvement. These six components are: *goal setting*, *interpersonal relations*, *role clarification*, *managerial grid*, and *problem solving*. We have stated in the initial part of this section what a puzzle game is, and it is rather evident how the problem solving activity is inherently a part of it. The cooperative feature of games seems to invest more than one team-building component (interpersonal relations, goal clarification), making it clear that puzzle cooperative games are a valid and viable option for team building activities.

2.2 Procedural Content Generation

"Procedural Content Generation" refers to any algorithmic computation with the purpose of building user-defined content with a "tunable" degree of randomness. "Content" is most of what is contained in a game: levels, maps, game rules, textures, stories, items, quests, music, weapons, vehicles, characters, and so on[6]. As stated by Smith [7], one of the first usage of PCG was in the tabletop board game *Dungeons & Dragons and Tunnels & Trolls*[10], allowing to generate dungeons by using random dice rolls. This example has the purpose to enforce the fact that PCG is not strictly related to a computer program, but is involved whenever some kind of agent (one that is able to follow a procedure) is generating content. On the other hand, it wouldn't be fair not to mention that the majority of PCG techniques are applied in the digital domain, and that in a variety of fields such as: videogames, computer renderings and, more generally, in visual [11] and non-visual arts[8]. Broadly speaking, PCG has influenced the Computational Creativity research area[9] and the scientific

[10]K. St. Andre. "Tunnels & Trolls". Flying Buffalo, 1975.

[11]K. Compton. "Practical Procedural Generation for Everyone". GDC, (2017)

academic community with the recent establishment of a mailing list[12], an IEEE CIS Task Force[13], a workshop[14] and a wiki[15] on the topic[10]. Inside the scope of this thesis, we will discuss PCG usage in videogames first, and then we will move to other kind of fields and medium that could be also used for videogame graphic and design purposes.

2.2.1 PCG in Videogames

Due to hardwawre (especially memory) constraints ath that time, in the early days of personal computer era it was required that developers find smart ways to craft interactive content. PCG techniques was firstly used to solve this issue, allowing to create content on the fly only when needed. One of the first videogames featuring PCG was *Beneath the Apple Manor*[16], a turn-based RPG, where the player, through a tile-based movement, has to traverse ten procedural generated dungeons (including enemies, secret doors, hidden traps and treasures) to accomplish to get the golden apple. Two years later *Rogue* [17] exploited the usage of PCG to deliver a similar experience, but with a more successful outcome, especially in college campuses[11]. The game was so inspirational that many ports and versions were released, while other variants set further steps for the genre: *Moria* [18] and *Hack* [19]. The Rogue variants became so widespread that now define a videogame genre, the *"Rogue-like"*: games, which generally feature turn-based exploration and combat in a high fantasy setting within a procedurally-generated dungeon and employ permadeath[11]. Apart from the emergence of rogue-like games (that by definition posses a procedural generation component), PCG was also used in other game genres: for example, the videogame *Elite*[20] was a space trading and combat simulator in which eight galaxies (each containing 256 planets) were generated at the beginning of each gameplay. Each planet had its own position, name, details and prices for the commodities. Because of the choice of a fixed seed number,

[12] `https://groups.google.com/forum/#!forum/proceduralcontent`

[13] `https://cis.ieee.org/getting-involved/30-technical-committees/376-games-technical-committee-task-forces`

[14] `http://pcgames.fdg2010.org/`

[15] `http://pcg.wikidot.com/articles`

[16] D. Worth. "Beneath Apple Manor". The Software Factory, Quality Software, (1978)

[17] M. Toy and G. Wichman. "Rogue". Epyx, 1980

[18] R. A. Koeneke. "Moria". 1983

[19] J. Fenlason and K. Woodland, M. Thome and J. Payne. "Hack". 1985

[20] D. Braben I. Bell. "Elite". Acornsoft, Firebird, Imagineer, 1984

the resulting generated "universe" was the same for each and every player: this apparent limitation (put by developers to hide the artificiality behind the game) does not avoid the rapturous reception of the consumers[12]. Other videogames of that period worth mentioning are: *Rescue on Fractalus*[21], an action first-person shooter that generates fractals in real-time, in order to resemble the environment on an alien planet, and *The Sentinel*[22] a puzzle videogame that could procedurally generate 10,000 landscapes. The Rogue and Elite heritage affects an increasingly growing market, by having given the birth to titles that heavily rely on PCG. In the 90's a spiritual successor of Moira, in a more modern key, was developed for PC: the action RPG *Diablo*[23]. Armors', weapons' and items' statistics were all randomly generated in addition to quests, sub-quests and whole levels, giving different random experiences in each playthrough[13][14]. Even though in the 90's the necessity of compressed content and expanding/generating it on the fly wasn't an issue anymore (due to modern computers), the usage of PCG was still significant, since it gave new allure to both covered and uncovered genres[15]. In the famous turn-based strategy game *Civilization IV*[24], players can experience different campaigns by setting the desired parameters in a map generator inside the game itself[16]: more deep map customization is given by the possibility of modifying game's XML data files or of editing game's Python scripts[17]. A few years later, the famous game designer Will Wright published its new game with PCG as a central component on top of a life simulation gameplay. The game was called *Spore*[25] and, thanks to an intuitive editor, the players could build their own creature animated through procedural-generation techniques[18]. *Boredrlands*[26] is an FPS action RPG that "uses a procedural process to generate its various guns in certain classes, such as handguns, shotguns, assault rifles, sniper rifles, and more, but with many variations of firing speed, reload speed, damage type and more" able to reproduce 17,750,000 of different weapon's variations[19]. On one the top positions of the best selling games of all time[20], there's one that uses PCG for world generation. *Minecraft*[27] is a survival sandbox that allows players to explore a world that is generated on the fly: rendered chunks of blocks are saved and loaded continuously, as the player explores the game world [21]. In the second half of 00's a huge comeback of the rogue-like genre is observable , mainly

[21]"Rescue On Fractalus!". Lucasfilm Games, 1984

[22]G. Crammond. "The Sentinel". Firebird, 1986

[23]Blizzard North. "Diablo". Blizzard Entertainment, 1997

[24]Firaxis Games. "Civilization IV". 2K Games, 2005

[25]Maxis. "Spore". Electronic Arts, 2008

[26]Gearbox Software. "Borderlands". 2K Games, 2009

[27]Mojang. "Minecraft". Mojang, Microsoft Studios, Sony Computer Entertainment, 2011

caused by the indie[28] development resurgence, due to the online game distribution bypassing the limitation of retail copies[22]. They share the same feature as their predecessors (so PCG is still used for generating content) but with a metagame[29] component, whereby the permadeath during the playthrought is rewarding: in this sense, players can unlock the possibility to use a new character or to find new items and monsters in the PCG levels whenever they start a new game[24]. Major titles of this new rogue-like wave are: *Spelunky*[30] (platform), *The Binding of Isaac*[31], *Crypt of the NecroDancer*[32](rhytm game) and *Slay the Spire*[33] (deck building game). Finally, it is worth to mention *No Man's Sky*[34]: despite a launch below the general expectations, it was able, after successive free updates, to give an actually enjoyable experience[25], offering 18 quintillion of procedurally-generated plantes with their biomes[26].

2.2.2 PCG in other fields

Procedural Generation has been also used in other professional fields other than videogames so extendedly that new software applications ad tools have been developed. *SpeedTree*[35] is a middleware software used for generation and animation of trees and grass employed in videogames (also AAA titles[36]) and in visual effects (including Hollywood awarded films). *Cityengine*[37] can procedurally generate 3D urban environment models on a large scales: it has been used in the videogame and movie industry and in urban, urban design and geodesign studies. Lastly *Massive Software*[38] was developed initially to render high scale battles in Peter Jackson's

[28]Individual or small-teams companies, focused on a specific game with low economic availability compared to big videogame development studios.

[29]"Metagame, or game about the game, is any approach to a game that transcends or operates outside of the prescribed rules of the game, uses external factors to affect the game, or goes beyond the supposed limits or environment set by the game"[23].

[30]Mossmouth, LLC. "Spelunky". Mossmouth, Microsoft Studios, 2008

[31]E. McMillen. "The Binding of Isaac". F. Himsl, E. McMillen, 2011

[32]Brace Yourself Games. "Crypt of the NecroDancer". Brace Yourself Games, Klei Entertainment, 2015

[33]MegaCrit. "Slay the Spire". Humble Bundle, 2019

[34]Hello Games. "No Man's Sky". Hello Games, 2016

[35]Interactive Data Visualization, Inc.

[36]`https://en.wikipedia.org/wiki/AAA_(video_game_industry)`

[37]Esri R&D Center Zurich

[38]Stephen Regelous

The Lord Of The Rings film trilogy and was used later in many important films. This software has the ability to render huge crowd simulations with automatically generated animations and behaviours.

2.2.3 PCG motivations

From previous subsection we can infer some glimpse of why PCG should, or can be, considered in a videogame. Now we are going to provide some of the reasons why PCG is used today, firstly in order to make clear the focus of this thesis work, and secondly why this technique is nowadays used especially for videogames. Evidently, some of these could be seen as a list of features that a game can have in its design (and for some of these it is actually the case), but it's obvious that this features can easily become valid motivation to use PCG.

- **Avoiding the expense of creating game content:** creating manually content in a game is expensive, slow[6] and in some cases "not-scalable" for large number of players[27]. Using procedural techniques is an alternative way to create content without putting a large burden on (or totally removing it from) designer and artists[1]. Replacing their work with smart and well crafted algorithms, is a production advantage that could bring to a faster, cheaper and still high-quality final product[6]. At the end of the generation process, it is desirable to have artists, game designers or a good crafted evaluation function, in order to assess the quality of the final product[1].

- **Replayability:** the game varies its content (necessary or optional[39]) in each playthrough, creating potentially "truly endless games"[10]. Th continuous variation of the game experience and the resulting unpredictability of playing, can provide incentives for the user: all of this is achievable with PCG.[7]

- **Personalizing experiences:** with some PCG algorithms it is also possible to align game content with the player's tastes. Using player modelling (e.g. model its response to individual game elements using neural networks) we can create a game experience that tries to maximize player enjoyment[6].

- **Augmenting human creativity:** the purpose of PCG is not to fully replace humans, but rather to augment their creativity[7]. Other than providing a (time and money saving) tool, PCG can increase the creativity of content

[39]Necessary content is required by the players to progress in the game (e.g. dungeons that need to be traversed), whereas optional content is the one that the player can choose to avoid (e.g. houses that can be entered or ignored)."[10]

creators, often offering unexpected and valid solutions that can inspire in their own future creations[6].

Before giving my own motivations for having used PCG in the present project, it is important to state that there is no secret sauce in PCG today. The task is rather difficult, starting with the representation choice of game content, up to the evaluation functions (used in "search based" and "generate and test" techniques that will be introduced later), both of which are game dependent. Most of the existing PCG algorithms are tailored for a single game content (except some few cases, e.g. middlewares such as SpeedTree) and no plug and play/reusable solution exists nowadays[28]. This leaves a lot of space for research, and goals with respective challenges can be found in Togelius et al[28].

As far as our project is concerned, we want to realize a generator that procedurally creates content for the proposed game. Our motivation to develop such generator is avoiding the manual creation of levels, and augmenting designer abilities to better polish the final product, by offering a supporting tool that (other than giving the possibility to play a prototype version of the game) analyzes in a deep way the resulting content and simulates possible outcomes and player behaviours. All in all, our final goal with this thesis project is not to replace the game designers, but rather simplifying their job with a general purpose tools for maze creations. We will not make any final decision in the parameters, as opposed as providing the designer with a supporting tool to explore configurations of parameters and states of the game, underlying statistics, and giving the possibility to maximize or minimize the content generated, with respect to certain defined metrics. Further notions regarding computer's role as an aid and support can be found in section 3.3 of this thesis.

2.3 PCG classification and approaches

In this section we are going to try to briefly classify the possible approaches to PCG, with a final specification about the one that has been used for this thesis work: all the content of this section owes much to Togelius et al[10].

- **Online versus offline:** The first distinction is whether the content generation is performed offline during game development, or online during the runtime of the game; the latter requires a predictable runtime, must be very fast and have a predictable quality.
- **Necessary versus optional content:** necessary content is the one crucial to game progression, while optional is the one that player can potentially skip.

Necessary content must be correct, namely it should not contain aberrations which makes it impossible for the player to progress.

- **Random seeds versus parameter vectors:** this choice defines the degree of randomness in our algorithm, with at one extreme a multidimensional vector of real valued parameters, and at the other a seed that randomly generates the parameter vector.

- **Stochastic versus deterministic generation:** by design, when generating content there is a variation at every run of the algorithm, versus having one fixed content every time the algorithm is run.

- ***Constructive* versus *generate-and-test*:** a constructive algorithm generates content whilst being aware about its correctness; thereby every operation, or sequence of operations, is ensured to never produce incorrect content. On the other hand, generate-and-test approach consists of two phases: a generation phase (where desired content is generated) and a test mechanism (that validates the content against some criteria). If the test fails the instance it is discarded, otherwise it is kept: the process of generation continues until some exit condition is met.

We will now move on to explaining *Search-based* PCG, which is a special case of generate-and-test approach (as mentioned in the last bullet of previous list) specifying its core features. Firstly, the content is evaluated again with some criteria, but also graded using a function that can be called "fitness", "evaluation" or "utility" function. The result of the function is generally a vector of real numbers that express the quality of the content. Secondly, the generation of new candidates depends also on the fitness values of previously evaluated instances: this means that previous candidates are considered in order to generate new ones with higher utility value and this is done by means of an optimization algorithm.

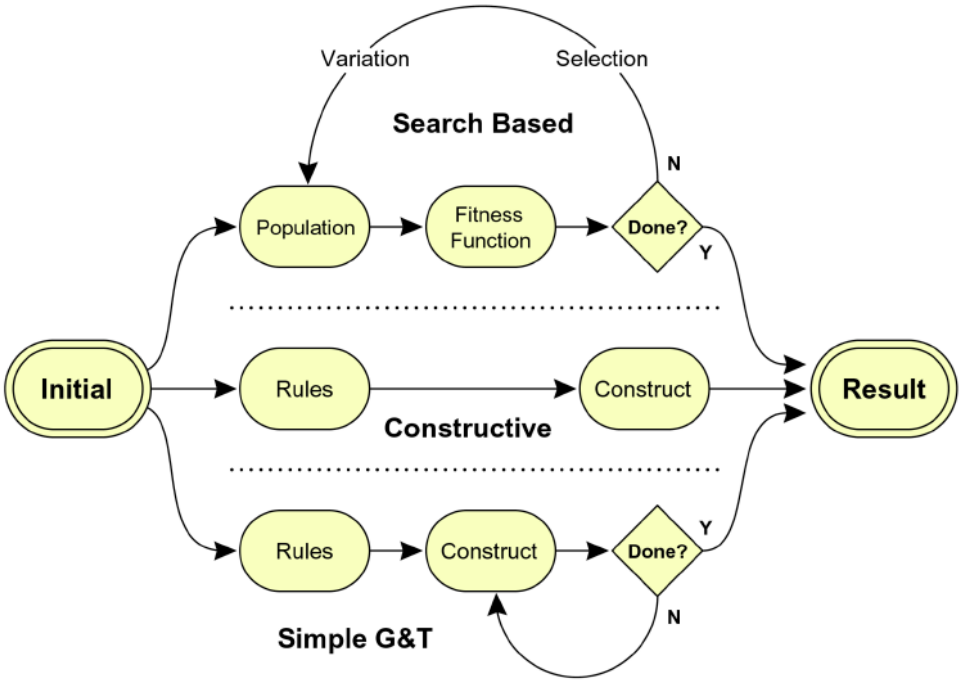


Figure 2.1: Three approaches to PCG: search-based, constructive and simple generate & test[10].

It is worth to mention that most of the search-based PCG application relies on *evolutionary algorithms*[40], where candidates have two parallel representations: one that is meaningful for the player in the usage of the actual content (phenotype e.g. a matrix representing a dungeon), while the other is handled by the evolutionary algorithm (genotype e.g. a list of suitably encoded rooms and corridors of the same dungeon). A starting random population is initialized and their genotypes ranked and (as it happens in animal evolution) the fittest candidates survive, while the other ones are discarded: at the next step, the population will be the surviving candidates plus new ones obtained by variation (mutation) of genotypes. Process of generation continues until a termination condition is met. Once this process ends, you will have on your hands the best-fitting genotypes, and all is left to do is to convert them into phenotypes, through a mapping function.

[40]Metaheuristic optimization algorithm that mimics natural evolution in looking for a best (fittest) solution (survivor).

The approach used in the thesis work is more centered on a generate-and-test approach (ensuring that the content satisfies some constraints e.g. end must be reachable) rather than a search-based one. That is because we are not focusing on a global optimum or on an approximation of it, but we want to offer a valid tool to search the range of possibilities that the generation could offer and one that gives back a resulting content made usable by constraint satisfaction. After that, we will also try to evaluate our content against a function, but this is not related to a "utility" of the content generated since only the designers can ensure it in our case and, as a counterexample, the target final behaviour could be also having a spread "utility" content. In conclusion, this is highly subjective and our work is to give to designers a tool to search the space and see which content, generated through parameters and having some statistics, constitutes their utility. Further information about this subject will be found in Chapter 4.

2.4 PCG in cooperative puzzle games

Cooperative puzzle games are an affirmed genre but the use of PCG to generate in-game content for this kind of games is very infrequent. The most known title is the above-mentioned *Keep Talking and Nobody Explodes* that sees two players in the challenge of defusing a bomb: one player has a manual explaining how to defuse the bomb while the second player must defuse it before the timer reaches zero. Defusing the bomb means solving puzzles and riddles on it generated procedurally by the game. In the scientific research a paper proposed by Arkel et al.[29] developed a cooperative puzzle-platform game using procedurally generative grammars to layout the levels and its puzzles exploiting game design patterns. Apart from these two examples, the usage of PCG in cooperative games genre is rather uncommon, mainly since generating levels for collaboration is more challenging because of the forced mutual benefits of the cooperation that puts an extra constraint on the design space[29]: so when considering a subset of games such as puzzles, the process can be only more difficult and challenging.

2.5 Procedural map generation

In this section we want to dissect the topology and the characteristics of PCG applied on maps defining the design of the game level. Most of the maps generated by PCG algorithms are grid-based: that is a n-dimensional space that contains a collection of elementary items called cells, interpreted as locations on the map.

The grid has an inherent topology notion where cells are connected with their neighbour cells allowing player movement. In general, every cell of the grid contains a binary (binary maps) or height values (heightmaps) that determines respectively whether to place an obstacles and the height of a cell: this information will be used at the end to render the game level. The formalism behind the grid-like map representation is the the *graph theory*. It is beyond the focus of this thesis to give a mathematical and complete description of graph theory concepts: rather, we are interested in explain what graphs are in the most synthetic and intuitive way, since they are a counterpart of grid representation and will be of great importance later on. Finally we will list some of the most used techniques and algorithms to procedurally generate grid-like maps (both with binary and height values) without considering generation of dungeon maps that are beyond the scope of this thesis.

2.5.1 Graph Theory

A graph is a triple $G = (V, E, \phi)$:

- A set of vertexes (or nodes): V.
- A set of edges that connects two nodes: E.
- An *incidence function*[41] that states how every edge is connected to a couple of unordered vertices: $\phi : E \to \{\{x, y\} | (x, y) \in V^2 \wedge x \neq y\}$.

Graph can be directed or undirected depending on the importance of the couple order: in our case, we always consider undirected graphs, so the codomain of the incident function will be unordered pairs of nodes, i.e. $\{x, y\}$ and $\{y, x\}$ are the same couples and they define the same edge.
A *weighted graph* is a graph where every edge has a weight associated to it (i.e. a number): we will consider mostly unweighted graphs unless it is clearly specified.
In a given undirected graph G, two vertices $v_i, v_j \in V$ are *connected* if it exists a path (list of subsequent edges) connecting them.
G is an undirected *connected graph* if $\forall v_i, v_j \in V : v_i, v_j$ are connected.
A *subgraph* is defined by the triple $G' = (V' \subseteq V, E' \subseteq E, \phi')$ where the new incidence function ϕ' is a restriction of ϕ. A spanning tree is an acyclic connected subgraph of a G where $V = V'$ i.e. the nodes in the subgraph G' are the same of G and every node is reachable from the other and vice-versa.

[41] In our formal definition, the incident function does not allow self-loops: this is not true in general but this is an additional constraint that better fits our map "shape" seen as a grid of graph nodes

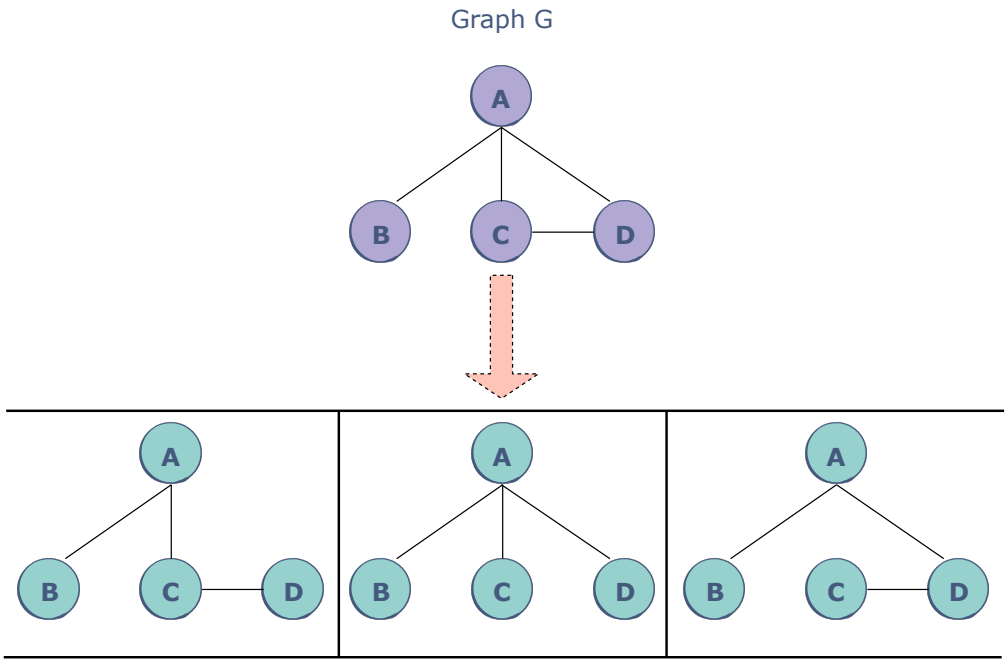


Figure 2.2: An undirected graph with all its possible spanning trees below[42].

Any grid-like map can always be seen as a graph allowing the usage of algorithms and access to well-studied formalism for its generation and exploration.

[42] https://www.educative.io/edpresso/what-is-a-spanning-tree

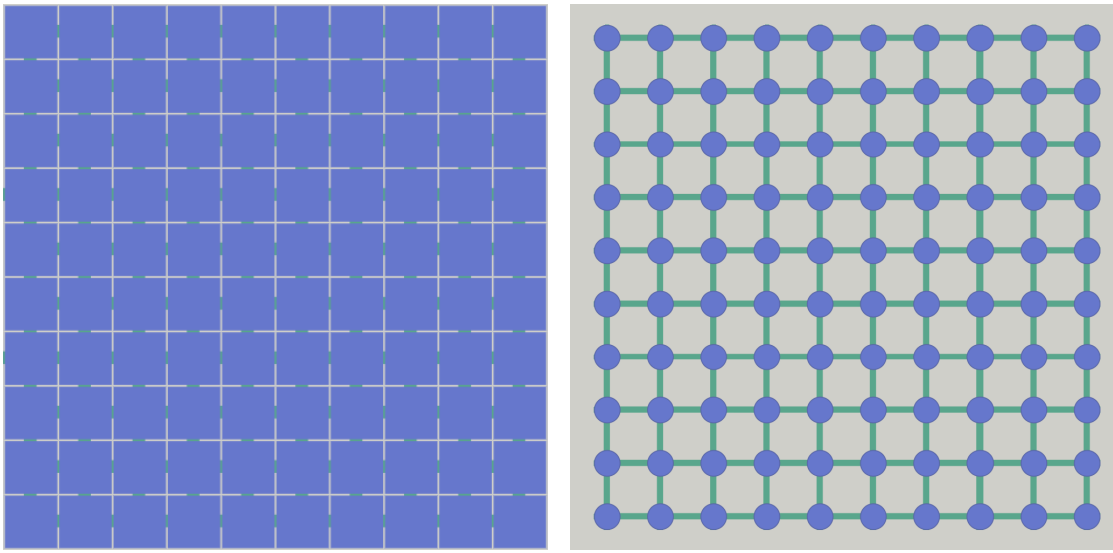


Figure 2.3: On the left we can see a 10x10 grid with every node accessible while on the right its counterpart: a graph representation where every node (in blue) is connected with the orthogonal adjacent nodes through the edges (in green)[43].

2.5.2 Maze generation algorithms

Graphs, and consequently grids, are the best and most common representations for 2D maze games. Generation of mazes has always been an argument of research and studies in the scientific field[30, 31, 32, 33] and there is a variety of techniques and approaches applicable. The common usage of the term "maze" is referred mostly to a particular kind:*perfect* mazes. Perfect mazes are very inflated and by far the most studied in the literature[30, 31, 32, 33]. A maze is perfect whenever in the graph defining the maze there are no cycles, no inaccessible areas and from every cell/node there is only one path to any other cell/node: this means that given a maze problem, in a perfect maze it exists only one exact solution (i.e. path) from the start to the exit. All the properties of a perfect maze are satisfied by a spanning tree in the graph representation so that whenever we find a spanning tree on a grid of cells, we obtain a perfect maze. Is not surprising that most of the algorithms to build perfect mazes comes from the graph theory field.

[43] `https://www.redblobgames.com/pathfinding/grids/graphs.html`

Prim's algorithm is one of the best known algorithms in maze generation and stems from graph theory. It is a greedy algorithm that produces a minimal spanning tree for weighted undirected graphs and this in turn can be directly converted into maze. For maze generation usually the graph is undirected and unweighted: random node/edge selection policy is used instead of the greedy one [30, 31]. Using partitions as obstacles[44] to layout the maze, the algorithm is:

Algorithm 1 Prim's algorithm

Insert partitions around all cells;
Initialise an empty frontier set f;
Mark one random cell as visited and insert all its neighbours in f;

while f is not empty **do**
  Remove randomly a cell c from f;
  Find all the neighbour cells N_c of c;
  Find all the neighbour cells $N_{cv} \subseteq N_c$ of c that has been visited visited;
  Sample randomly a cell c' from N_{cv};
  Set c' as visited (i.e. in the path);
  Remove the partition between c and c';
  Insert $N_{c'\bar{v}} = N_{c'} - N_{c'v}$ cells into the frontier f;
  Remove c' from f;
end while

For its random policy this is also called *randomized Prim's algorithm*[33]. Thanks to its simplicity, *recursive backtracker* is another well known algorithm in the literature: it builds a maze using the *depth-first search* algorithm[30, 31] for tree and graph search. This algorithm is used also for maze generation since it produces a spanning tree i.e. a perfect maze. It requires the usage of a *stack*[45]: a well known *abstract data type* in Computer Science. Using again partitions as obstacles to layout the maze, the algorithm would be:

[44]Obstacles inside a maze can be walls (i.e. removing the cell and its incident edges) or partitions (i.e. removing an edge between two adjacent cells).

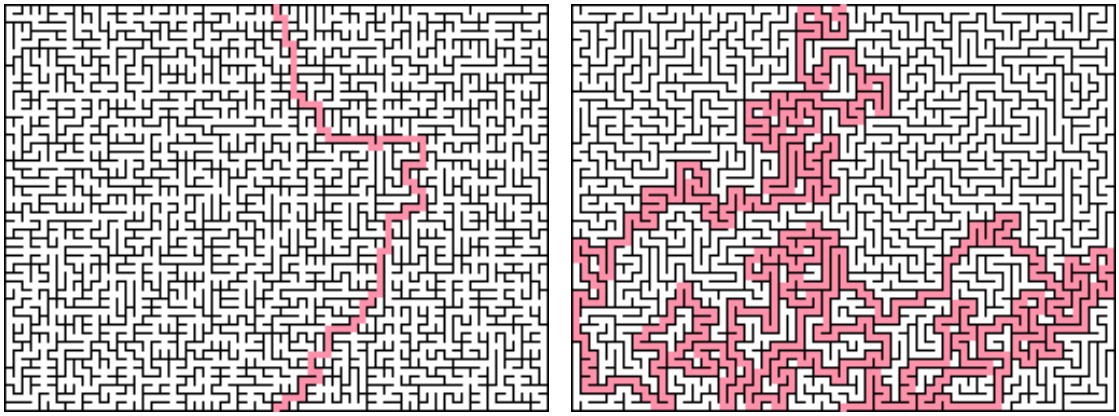
[45]Is a *LIFO* (last in, first out) structure with two one basic operations: *push* that adds an element to the structure and *pop* that removes the most recently added element.

Algorithm 2 Recursive Backtracker algorithm

```
Insert partitions around all cells;
Initialise a stack;
push a randomly chosen cell into the stack and assign it to c (i.e. current cell);

while stack is not empty do
    if |N_cv̄| > 0 then          ▷ if the current cell has any unvisited neighbours

        Randomly choose one cell c' from N_cv̄;
        Add c' to the stack;                                  ▷ c' is the next cell
        Remove the partition between c and c';
        c = c';                                ▷ set the next cell as the current cell
    else
        c = stack.pop(); ▷ make the current cell as the latest added to the stack
    end if
end while
```

The difference between the two algorithms stems from the selection policy of the cell to expand at the beginning of the *while* loop: given an iteration, the neighbor selection of the partition to remove is random. A good comparison is done by Pullen[34] where both of the algorithms are *bias free*[46] and Prim's produces more dead ends w.r.t recursive backtracker: in addition Prim's produces solutions that are shorter and faster to solve w.r.t recursive backtracker that tends to produce longer ones[32].

[46] A maze generation algorithm is bias free when it tends to treat equally all directions and sides of the maze.



(a) Maze generated using Prim's algorithm.

(b) Maze generated using recursive backtracker algorithm.

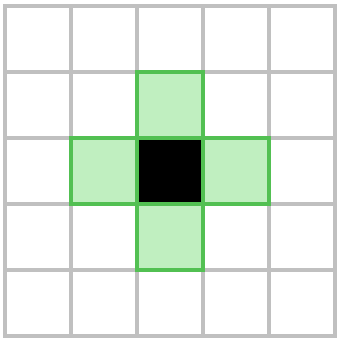
Figure 2.4: The start cell and end cell are respectively on the bottom center and on the top center. The pink line shows the solving path underlying the difference between the two generation algorithms[34].

The drawback of both algorithms is that they produce *non-uniform* mazes: a maze is uniform when, given a graph representation, all the possible mazes (i.e. spanning tree) are equally likely in the final result. Prim's doesn't sample mazes (and spanning trees) randomly, while recursive backtracker never produces some of the possible mazes. This "drawback" depends on the kind of applications involving mazes (or also spanning tree generation): uniformity can be a wanted or an unwanted feature. Nonetheless, both these algorithms are very well known and used for perfect maze generation thanks to their simplicity and the very different results that they produce.

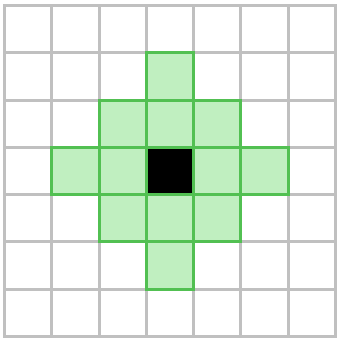
2.5.3 Cellular Automata

A *cellular automata* (CA) is a grid of cells each with a binary state, that evolve at discrete time steps changing their state according to the neighboring cells[35]. In the early 1950s Von Neumann incorporated CA in one of his works becoming one of the first to consider such a model. The best-known CA is *Conway's game of life* discovered by the homonym mathematician in 1970 to the point that a typology called *life-like cellular automata* takes inspiration from its discovery[36]. Their usage ranges from biology[37], chemistry[38] to computer applications[39] and the latest years has also been used as a PCG technique to model environmental systems[40]. More recently CA techniques have been used by Johnson et al.[41]

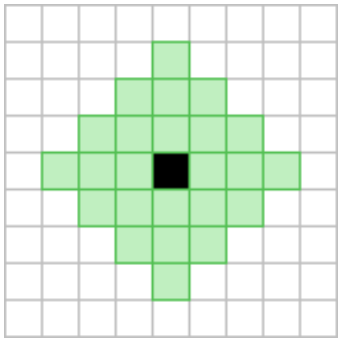
to layout an infinite cave-like 2D map on a grid. Crucial for the behaviour of the CA is the neighborhood choice: that is, which adjacent cells are to be considered "neighbors" to a given cell. The most used neighbourhoods are the *Von Neumann* and the *Moore* one.



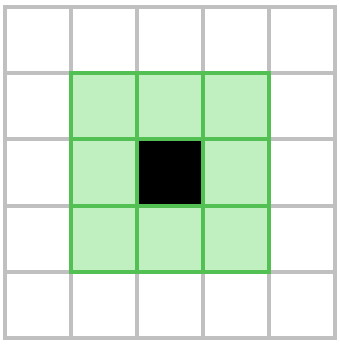
(a) The Von Neumann neighbourhood (green) of a single cell (black) of range 1



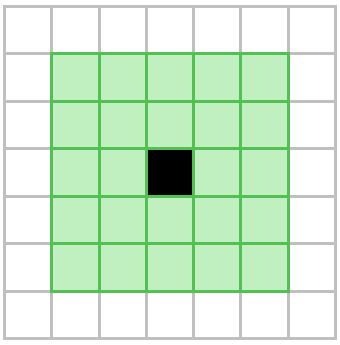
(b) The Von Neumann neighbourhood of range 2



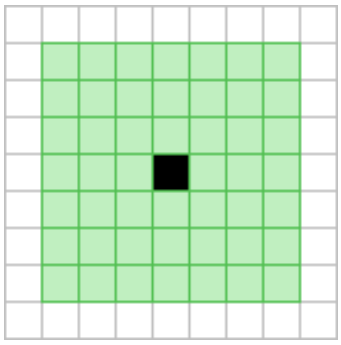
(c) The Von Neumann neighbourhood of range 3



(d) The Moore neighbourhood (green) of a single cell (black) of range 1



(b) The Moore neighbourhood of range 2



(c) The Moore neighbourhood of range 3

Figure 2.5: The most common neighbourhoods with different ranges[36]

Johnson et al. used Moore neighbourhood and defined four input parameters of the CA generator algorithm:

- r**:** initial percentage of walls
- n**:** smoothing iterations

- T**:** neighborhood threshold that defines a wall

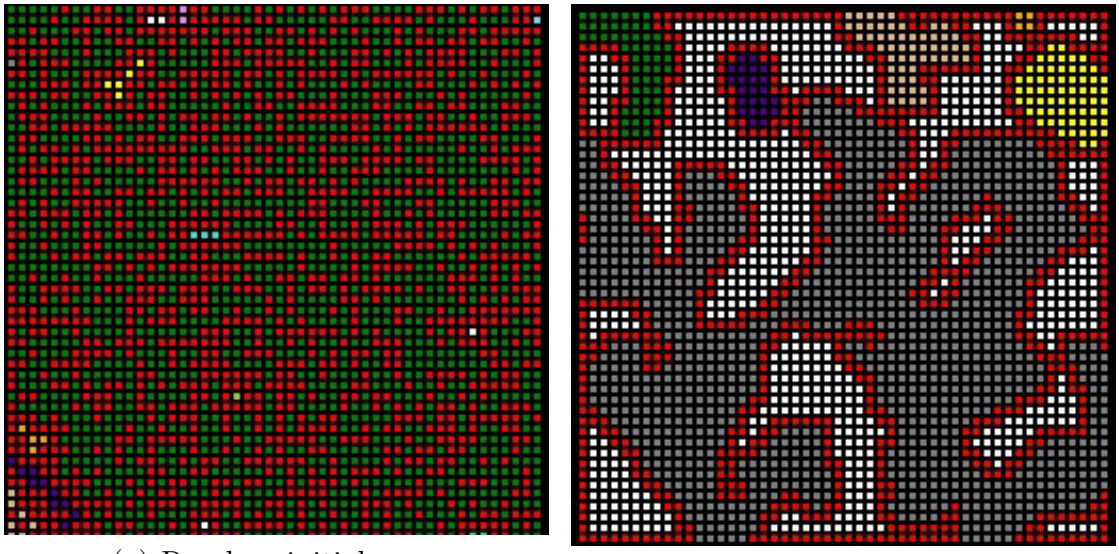
- M**:** size of the Moore neighborhood

As it usually is for a CA, the algorithm processes a 2D grid map where each cell can be in one of two states: floor and wall[47]. The generation proceeds as follows:

Algorithm 3 Cellular automata for PCG

```
Set every cell of the map as room;
Randomly set every cell of the map to wall with r probability;
for n times do
    for every cell c of the map do
        if Moore Neighbourhood of c of size M ≥ T then
            Set c as wall;
        else
            Set c as floor;
        end if
    end for
end for
```

[47]In Johnson et al. three states are defined: floor, rock and wall. A wall cell is special case of rock cell that has at least one room as neighbour. Without loss of generality we consider only two states (floor and wall) that are the baseline for a CA approach.



(a) Random initial map. (b) Final map.

Figure 2.6: CA generation process in Johnson et al.[41] with parameters: $r = 0.5$, $n = 4$, $M = 1$, $T = 5$. Walls with at least one neighbour floor cell are represented in red, while isolated walls are white. The floor clusters are highlighted with different colors.

Simplicity and intuitiveness is a pro for this generation alghorithm. On the other hand it is difficult to predict the effects of a single parameter on a general instance of the generation process, since each parameter affects many features of the resulting maps[42]. Finally, is not possible to embed precise gameplay requirements into the generator (e.g. adjacency of maps in Johnson et al.) tuning the input parametrs unless with a trial and error approach.

2.5.4 Terrain generator

These kind of generators try to mimic the appearance of natural terrain producing a 2D heightmap as output. This is achieved by trying to mimic *fractals*[48] since terrain, in real-life, is a fractal itself. The terrain heights on the mesh will correspond to

[48]Self-similar structures: if magnified, subsets of a given object look like (or identical to) the whole and to each other. One practical example can be the jagged edge of a broken little piece of a rock: it has the same irregularities as the contour of the same rocky mountain on a distant horizon.

numerical values contained in a 2D array (heightmap) where heights of each point is initially set to zero. The mainstream algorithms for terrain generations are:

- *Diamond-square[43]:* it is an improved version of the *midpoint displacement* algorithm by Miller[44] that reduces its artifacts by disallowing low-averaged heightmap values. In order to obtain this result, initialize corners of the heightmap with random values. Then, set the center of the heightmap to the average plus some random value: corners trace a square. At this point, compute the average (plus some random value) on the midpoints of the horizontal and vertical edges of already computed points: those vertices layout a diamond shape. Finally, repeat the last two operations recursively, taking smaller chunks of maps while lowering the random value at each iteration: continue until you reach the closure (3x3 heightmap).

- *Perlin noise[45]:* it is a gradient coherent noise function oscillating between 0 and 1 (considering the bi-dimensional case) developed by Ken Perlin. It exploits randomly chosen gradient vectors to smooth out the difference value of adjacent points. Perlin noise function has an amplitude (that defines the magnitude of the output value) and a frequency (that says how fast the output values oscillates). Terrain generation is achieved combining multiple Perlin noise functions at different amplitudes and frequencies. The common choice are octaves: functions of double frequency but half the amplitude. Adding recursively more octaves puts some low amplitude and high frequency components that give a natural looking fractal. Ken Perlin improved this noise developing and patenting his newer, faster and more advantageous noise called *Simplex noise*[46]: the generation process for terrains does not change, but uses this more performant noise function instead.

Chapter 3

Collaborative Maze Project

In this chapter we can finally focus on the thesis work: in particular, we will cover the game project (product of a collaboration with the start-up *AnotheReality*[1]) and the main core features of the work done. To allow a better reading of the dissertation, we will now define our nomenclatures and assumptions. Finally, we will categorize our tool and state the objectives we want to achieve with it.

3.1 The Maze problem

Humans have always been fascinated by mazes: some evidence of this can be found in myth and ancient history. Scientific research have tested capabilities of maze traversing of laboratory animals[47] and artificial intelligence robots[48, 49]. Since the second half of the nineteenth century, algorithms for maze exploration have been studied thanks to Graph Theory that helps in properly formalizing the problem[50, 51]. In section 2.5, we have seen that a maze can be considered as a n-dimensional grid space and parallelly a graph. In the research, the maze problem (that defines our gameplay) consists of a starting point from a location (inside the maze) to reach a goal location. The agent (aka player) is able to move inside it in any of the neighbour locations, provided that the next one is not occupied by a wall or separated by a partition. So describing a maze (and its structure) means describing the displacement of obstacles (walls or partitions), or of the walkable locations instead. One of the two must be directly or indirectly defined. A *direct representation* of a maze challenge (or problem) is a grid describing the layout of the maze/map, by defining the nature of every cell: walkable, obstacles,

[1] `https://www.anothereality.io/`

start and end cells. This can be easily implemented using a matrix of boolean or character variables. Mazes do not stop only to a square grid representation: a wider classification exists differentiated by seven points, and many kinds and possibilities have been explored[34]; but listing precisely every kind of possible maze structure that we could find is outside of our scope.

All the maps considered in our thesis work represent the actual game level (i.e. direct representation) that holds a maze challenge in two dimension, in which you can move the player in the usual euclidean space of a rectangular grid of square cells. Indirect representation was not chosen as a first option because it does not fit the problem very well, since the game is strongly cell biased and not related to specific structures such as long walls, rooms, and special tiles (specific configuration of cells).

From now on, when we refer to "maze" considering its structure, we will always refer to the topology; by that we mean the map (level design) and specifically the value of the cells contained in the matrix of characters that set up the maze challenge. Any other contexts in which we may use the term "maze", are going to refer to the problem or challenge given to the player of starting from a location and find a path to a goal location.

3.2 The Game Concept

In this section we will make a brief and concise list of the core ideas behind the game proposed by AnotheReality. The game is a *virtual reality*(VR) game, designed as a collaborative puzzle to be used during team-building events hosted by companies, while AnotheReality will make all the hardware to play with available. The puzzle consist of a maze challenge and the collaborative component is an *asymmetric multiplayer* gameplay[2]. The game main *mechanics* and *dynamics*[3] are summarized in the following list:

- Using a VR headset, one player will find him/herself inside the maze, while the other teammates (aka the "helpers") try to guide him/her through the maze in order to reach the end, while being outside of the game.
- Only one player is in the VR experience while the helpers (one or more) do not have any vision of what the player sees.

[2]Players (as a team or not) have different roles, abilities and gameplay experiences.

[3]See “MDA: A Formal Approach to Game Design and Game Research” for further information.

- Helpers have a set of maps displaying some maze challenges. In each one there is a start (blue cell) and end (green cell) location.

- Only one of the map in the set is the one the player in VR is actually playing, while the other ones are fake maps (alias maps). When we talk of "alias map" we intend a map with walls, rooms, one start and one end cell that tries to confuse the player during the real map finding. We refer to this set of alias maps shown to the helpers as *alias set*.

- There is a correspondence between cells in the maps displayed to the helpers, and the rendered rooms in the VR experience. These are orthogonally interconnected by corridors (as long as the cell are squared).

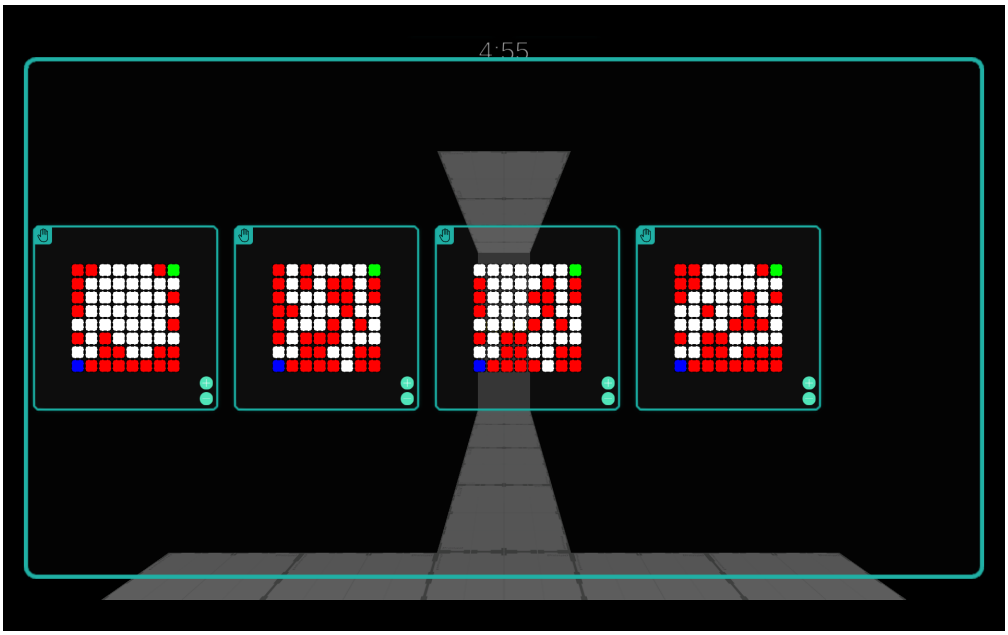


Figure 3.1: A screenshot of a possible set of maps displayed to the helpers during the game: one of them is the one that the VR player is into.

- Assuming that the player in the VR experience always starts inside the maze on the blue cell facing the Y positive axis of the grid, the teammates has to guide the player through the maze before a countdown timer goes to zero.

- The player inside the maze is in one cell of the grid but he/she has no vision of neighbour cells because of a *fog of war* or sliding doors that cut the vision in corridors taking to adjacent cells.

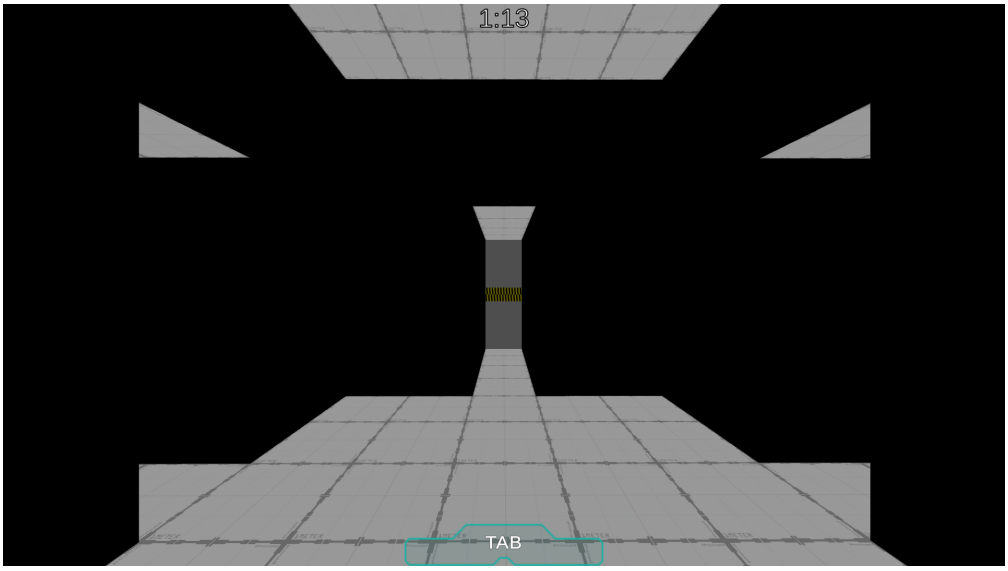


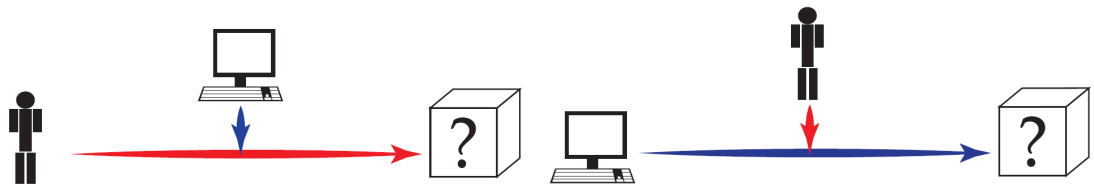
Figure 3.2: A screenshot of the prototype of the game. In our prototype, sliding doors where chosen to cut the vision from the current room/cell to the neighbouring rooms/cells.

- The helpers have to suggest which direction the player should go next according to the information they have. Meanwhile, the player in the VR experience has to communicate constantly which new rooms or obstacles he/she encounters, so that helpers can rule out fake maps present in the set.
- The kind of rooms the player can encounter are: rooms (i.e. free rooms), wall rooms and borders. Entering free rooms or finding maps border does not affect the player, while entering a wall room will imply a time penalty, reducing the available time.
- The goal of the cooperation is to rule out all the fake maps as soon as possible, in order to set the real map apart from the ones available to the helpers. Having only one map left in the set, means that helpers have found the real one and guiding the player inside the maze to the green cell is going to be straightforward.

As far as the *aesthetic* is concerned, the setting (following the company's suggestion) should be sci-fi: a drone (played by the player in VR) must escape a factory before the exit closes (when the timer reaches zero). The aesthetic was not a concern during our thesis work since our tasks were to dissect the gameplay of the given game (thanks to tools that handle generation, suggestion and analysis) and to realize a simple prototype of the game concept with the core aforementioned mechanics.

3.3 Computer aided design

Computer programs always aim to solve a specific task and their exponential usage has expanded also in professional fields by solving engineering problems, supporting creative design process or by automating tedious human tasks. With the goal of simplifying the user's life, Computer aided design (CAD) employs the usage of a computer system (specifically a software) to aid the designer during its design process, that is the act of creating, modifying, analyzing and optimizing a design[4][52]. According to Lubart, the most ambitious role for computers is to become a "colleague" for the designer, when random or semi-random search mechanisms are involved by the computer to generate novel, unconventional ideas[53]. Specifically concerning the usage of PCG, Liapis et al[54] discussed software developed to assists the human designer, referring to it as *mixed-initiative content creation* (MICG). To be of any use, a mixed-initiative PCG program has as a baseline the need of human input during the generation process: both human and computer take the initiative to achieve the final goal, that is the PCG in this setting (MICG). The degree of initiative arising from one of the two parts, will bring about two possible scenarios: at one extreme you will find CAD where "the human creator has an idea for a design, requiring the computer to allow for an easy and intuitive way to realize this idea"; at the other end of the scale there will be *interactive evolution* where the computer creates an initial content and continuously refines it following the human preferences.



(a) Computer-aided design: Humans have the idea, computers support their creative process.

(b) Interactive evolution: Computer create content, humans guide it to create content they prefer.

Figure 3.3: Two types of mixed-initiative design according to Liapis et al.[54].

In Liapis et al. the authors both describes and surveyed many academic tool developed for MICG, and in particular the "Sentient Sketchbook: Computer-Aided

[4]"Design" is intended as "the act of devising an original solution to a problem by a combination of principles, resources and products in design"

Game Level Authoring"[55] was taken as main inspiration for this thesis project, with special regard to the accounting for a novelty feature in the CAD framework.

3.4 Final considerations

The reasons for using CAD listed by Sarcar et al.[52] share some purposes with the ones for using PCG (see section 2.2.3), mainly regarding the increase of productivity for designers and the possibility to get an high quality final product. Keeping this in mind, we want to finally state the intention of our project called *"Collaborative Maze Project"* (CMP). Our main objective is to develop a MICG software, in particular a CAD able to assist game and level designers of the company we are collaborating with to describe, play and inspect the videogame prototype described in section 3.2. As common for CAD tools, CMP is able to automate content generation and evaluate the content generated, in particular its feasibility and gameplay specifications. As usual for CAD softwares, CMP has an user interface that gives real-time feedbacks, providing with the possibility of tuning parameters and displaying useful information about them and other CAD components.
From now on, every time we talk about " alias challenge", "problem" or "game" inside the thesis work, we implicitly refer to the game proposed by the company and completely described in this chapter.

Chapter 4

Project implementation

In this chapter we will discuss extensively the work done with all the details and reasoning needed to justify our work. We will avoid showing directly the code but rather explain the structure and the algorithms behind the core parts of the thesis project. Our thesis goal is to realize a CAD tool for the videogame described in chapter 3.

Giving all the premises stated in the previous chapter, Collaborative Maze Project consists of three components:

- *Map Generator:* this is the first component shown at the first execution of the CAD. With this tool, the user (i.e. game or level designer) can create a 2D map for our game using some of the most used PCG algorithms for 2D map generation that fit our game setting (grid of free and wall cells). The result of this component is the real map and it is mandatory to generate it since the other two components depend on its existence.

- *Alias Challenge Generator:* optional CAD component where aliases of the real map are generated. Both similar and novelty maps are reported in addition to different possible best paths found by *state-space search*[56] and agents simulation. Of each different path, the decrease of remaining aliases is reported on a Cartesian chart.

- *Game Level:* the actual game to be played. Once the designer has chosen a real map in the Map Generator (and optionally its aliases), he/she can play a prototype of the videogame already discussed.

Collaborative Maze Project is completely realized in *Unity*[1]: one of the most

[1]Unity Technologies

used and well-known cross-platform game engines. This choice was done due to the ease of programming and prototyping (including VR experiences like in our case) that Unity offers; thanks to its continuous updates and libraries by community members, it is the perfect choice for this project.

4.1 The Map Generator

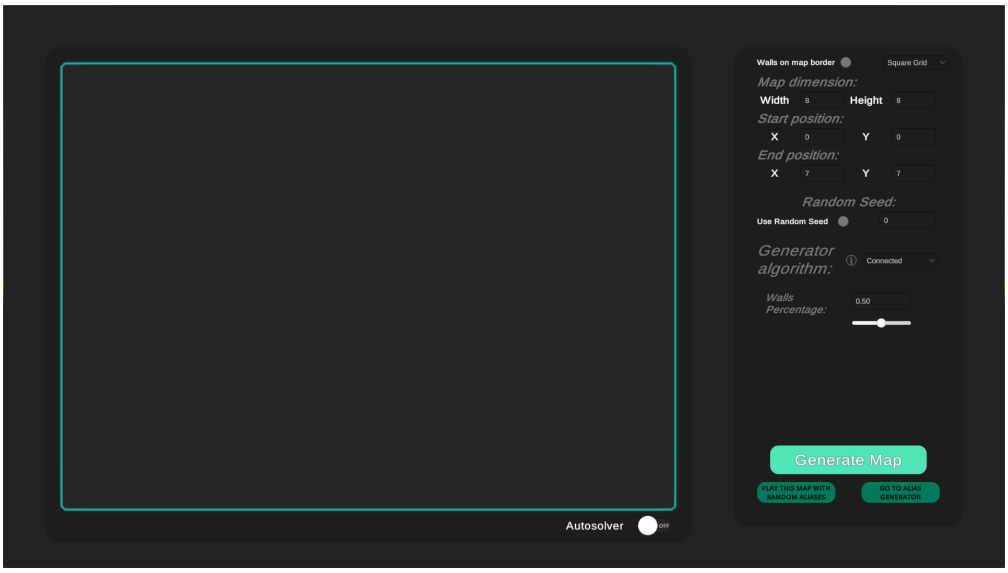


Figure 4.1: A screenshot of the Map Generator: on the left the map container (delimited by green water border) and on the right the parameter panel (in dark grey).

Map Generator (reported in the screenshot above) offers three different generators with all the selectable parameters (on the dark grey right panel) and a container for the map to be shown (delimited by the left green water border).

4.1.1 The Map Container

The map container displays some important statistics for the map (when present) on the bottom while there are some buttons on the bottom-right corner to zoom-in and out the map and save it (as a matrix of characters) on the PC in a .txt file format.

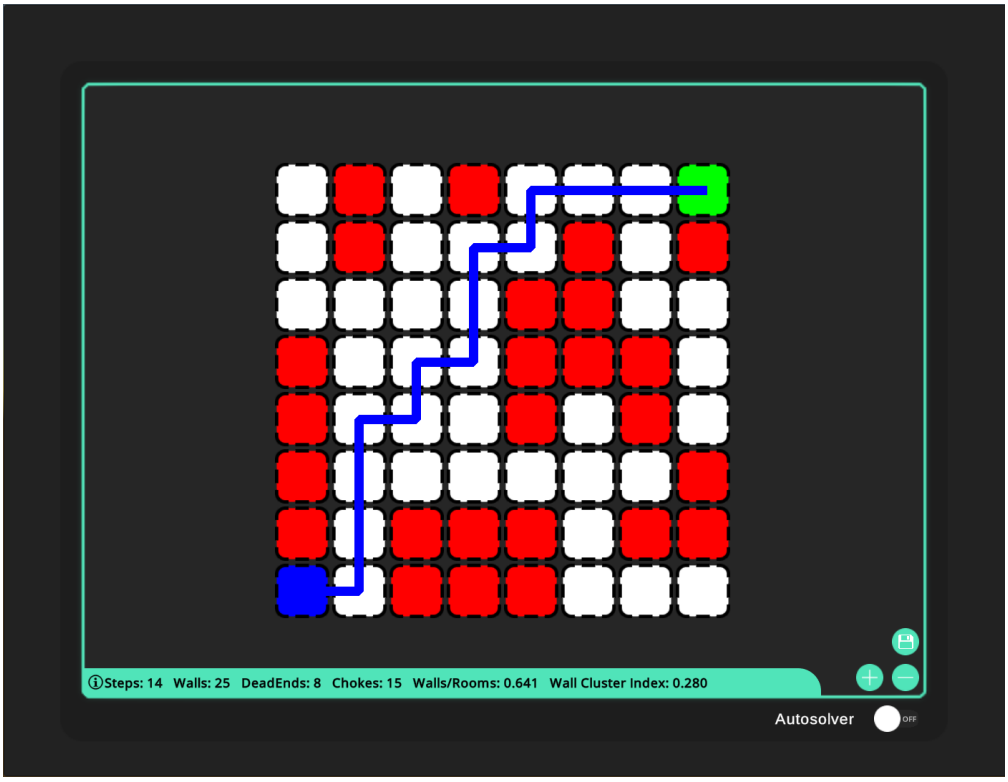


Figure 4.2: A screenshot of the map container: on the bottom-left map statistics while on the bottom-right the button to respectively save, zoom-in and zoom-out the map.

The info icon box in black shows (when the mouse is hover) important explanation of the statistics. The statistics displayed for the map in the Map Generator are:

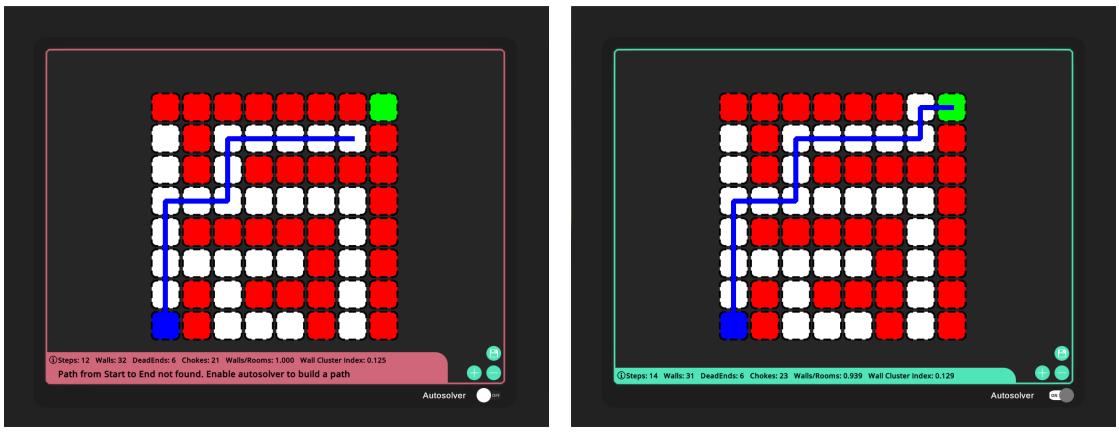
- *Steps:* total number of steps from start cell (blue) to end cell (green) following the optimal path[2].
- *Walls:* number of wall obstacles cells (red) in the map.
- *Dead-ends:* number of room cells with exactly one neighbour room.
- *Chokes:* number of room cells with exactly two neighbour rooms.
- *Walls/Rooms:* walls (red cells) count divided rooms (white cells) count. Values near to zero imply the prevalence of rooms wrt walls while the opposite for

[2]Optimal path (blue line) is computed using the well-known $A*$ algorithm. See *"Red Blob Games: Introduction to $A*$ algorithm"* for a synthetic and concise explanation on the shortest path computation in a square orthogonal grid setting.

values greater than one. If the value is exactly one, the number of walls and rooms is the same

- *Wall Cluster Index:* fraction of total wall cells that are in the biggest connected wall area. A value equal to 1 means that there's only one big wall cluster in the map.

On the bottom we found the toggle button for the *autosolver*. This feature allows to dig a path from the cell nearest to the end cell and reachable from the start location. The path carved follows the linear interpolation algorithms from orthogonal grid walking[3].



(a) Map generated but with unreachable end cell.

(b) Map regenerated after switching on the autosolver.

Figure 4.3: When enabled, autosolver allows to keep all the unusable maps during the generation process.

[3] `https://www.redblobgames.com/grids/line-drawing.html`

4.1.2 The Map Parameters Panel

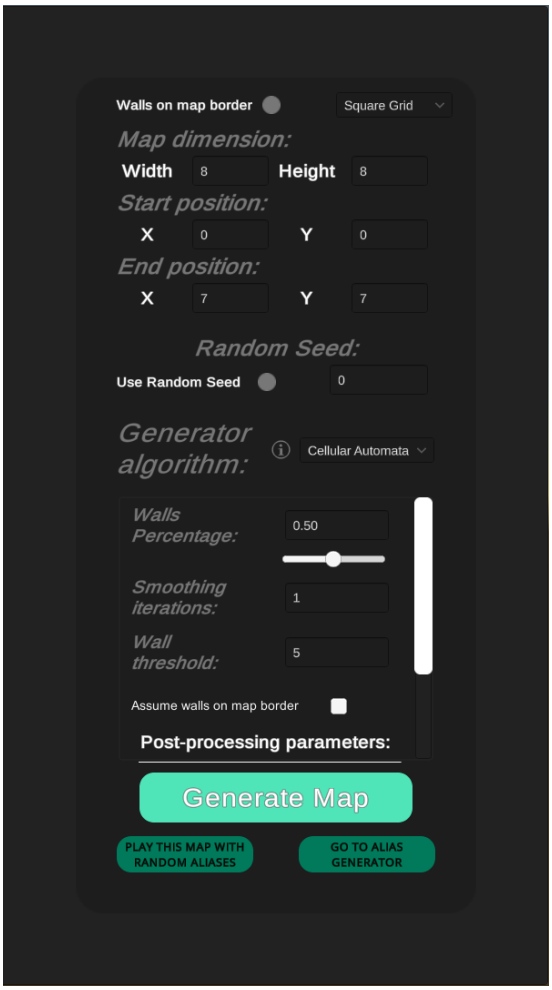


Figure 4.4: A screenshot of the parameters panel: on top of the "Generator Algorithm" text we have all the common parameters while on the right there is the drop-down menu to select the desired generator. This choice changes the parameters displayed down below (inside light grey border) that are specific for the generation algorithm.

This panel contains input fields that allow the designer to tune the generation process. The parameters common to all the generators are listed in the upper part and are:

- *Map dimension:* the width-height couple that specifies respectively the number of cells along the X and Y axes of the map.

- *Start position:* the coordinates of the start position for the player (blue cell).
- *End position:* the coordinates of the end position (green cell). When the player reaches the end position, the game ends.
- *Random Seed:* specify the random seed to be used in the generation process. When the toggle on the left is on, the seed is randomly chosen and displayed on the right input box, while when the toggle is off, the value contained on the right input box is used.

In our framework a map is *valid* when start and end cells positions are inside the map dimension and when it exists a path (so a set of free rooms) from start to the end cell i.e. the end (or equivalently the start) is not an isolated cell. In the second case the autosolver can help the designer to generate a valid map from the invalid map instance. PCG techniques can be selected through the drop-down menu. Directly on the left, a white info icon provides some information for the selected generation algorithms when the mouse is hovering the icon. Below the info icon (delimited by a light grey border) there is the panel used to display *specific parameters* for the PCG algorithm chosen: all this specific parameters relative to a given PCG algorithm will be discussed in their respective subsections.
Finally, at the bottom of the map parameters panel, there are three buttons: the first is used to generate the map when all the parameters specified (for a selected PCG technique) are valid. Last two buttons are used to play a random instance of a game level when a valid map is generated and to access the Alias Challenge Generator.

4.1.3 Connected Generator

Generation technique that produces a map where all free tiles are connected. The designer specifies a percentage of walls wrt the total number of cells in the map.

Specific parameters:

- *Walls Percentage (r):* walls percentage in the final map.

The pseudocode of this maze generation algorithm is the following[4]:

[4]The algorithm used is a slightly modified and adapted version of the map generation algorithm done by Sebastian Lague[57]

Algorithm 4 Connected generator algorithm

```
Insert all maps coordinates in a queue q;
Shuffle randomly q;
                                        ▷ Compute the number of walls to insert
wallsCnt = r * mapWidth * mapHeight;
i = 0;
currWallsCnt = 0;

while i < wallsCnt do
    currCoord = q.Dequeue();                    ▷ i.e. take a random coordinate
    q.Dequeue(currCoord);                                ▷ circular list behaviour
    Insert obstacle in map at coordinates currCoord;
    currWallsCnt = currWallsCnt + 1;

    if currCoord = startCoord or currCoord = endCoord or
¬IsMapFullyAccessible(map, currWallsCnt) then

        Remove the obstacle in map at coordinates currCoord;
        currWallsCnt = currWallsCnt − 1;
    end if

    i = i + 1;
end while
```

The algorithm tries to insert *wallsCnt* wall cells checking that, at every placement trial, every free cell is reachable. In order to do this, the number of room cells before and after the wall insertion are checked and if it is the same, it means that no rooms becomes isolated (unreachable from start position). This in the *IsMapFullyAccessible* function thanks to a *flood fill* algorithm.
It is important to note that when the walls percentage is near or equal to the unit, the algorithm does not create a map filled of only walls. This happens because the priority is to keep a fully connected map at every step: at every iteration the algorithm tries to insert a wall and if no more walls can be inserted (because it violates the connectivity) the trial fails. It is evident that when the maximum number of walls that keeps the map fully connected is reached, the map does not change: when this happens the resulting maze is a perfect one.

4.1.4 Cellular Automata

The cellular automata generator produces a cave-like map. We have already discussed its implementation in section 2.5.3 with all its main parameters. In our work we have defined an additional post-processing operation that, in turn, requires two specific parameters: all of them are listed as the lasts of the following list.

Specific parameters:

- *Walls Percentage (r):* initial percentage of walls in the map.
- *Iterations Number (n):* number of iterations in the CA algorithm.
- *Threshold Wall (T):* number wall neighbours above this value, it makes the tile a wall while under this value the tile is a room.
- *Border is a wall (brdWall):* if true, out of bounds of the map are considered as wall cells, rooms otherwise.
- *Room Threshold:* connected regions of room cells equal or below this threshold turns into walls.
- *Wall Threshold:* connected regions of wall cells equal or below this threshold turns into rooms.

A function called $GenerateCA(r, n, T, 1, brdWall)$ is a slightly modified version of 3 (where $G = 1$) that produces a map as an output. The modified version is because of $brdWall$ parameter that explicitly says how to consider out of bound cells during the CA-iterations.

Algorithm 5 Cellular automata algorithm

```
                                        ▷ Initialization and generation operations
map = GenerateCA(r, n, T,1, brdWall);

                                        ▷ Post-processing operations
for Every isolated region of free cells in map do
    if Number of free cells in region < Room Threshold then
        Turn the whole region in walls;
    end if
end for

for Every isolated region of wall cells in map do
    if Number of wall cells in region < Wall Threshold then
        Turn the whole region in rooms;
    end if
end for
```

4.1.5 Prim's algorithm

This algorithm tries to reproduce the same results of Prim's, but using walls as obstacles instead of partitions (see section 2.5.2 for further information). One versatile way to get a perfect maze using walls and Prim's, is using *grid colorings* discussed by Tulleken[58]. Grid coloring is a repeating pattern of K distinct colors. Two vectors u and v dictate how far the same color repeats on vertical and horizontal axes of the 2D grid: this defines the pattern area and so the maximum number of colors that can be specified, which is $N = u_x v_y - u_y v_x$.

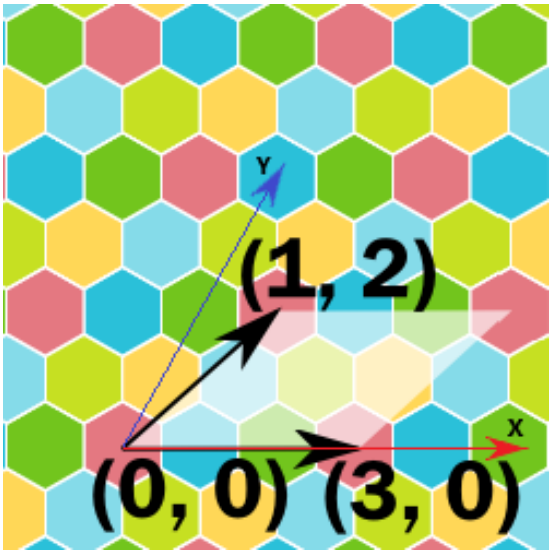


Figure 4.5: Grid colorings on a hexagonal grid in axial coordinates (red and dark blue axes). In this case $u = (3,0)$ and $v = 1,2$ allowing $N = 3 * 2 - 0 = 6$ possible colors[5].

Placing $K \leq N$ different colors on the area given by the two vectors, completely defines the pattern.
In order to use grid colorings for maze generation, we must first define a valid pattern, and after to which category of cells the K different colors belong: free, walls and pillar cells. Pillars are simply walls that cannot be replaced while walls can be carved and become rooms. Given a grid coloring, deciding which colour is a free, wall or pillar cell must satisfy the following conditions:

- Each room is adjacent to four walls.
- Each wall is adjacent to one or two rooms.
- If a wall is adjacent to a room, the room is adjacent to the wall and vice versa.

If all the condition are satisfied for a given grid coloring, we can then apply the generation algorithm.

Specific parameters:

- *Removing walls percentage (r):* specify the percentage of walls to delete in order to make the maze a non-perfect one.

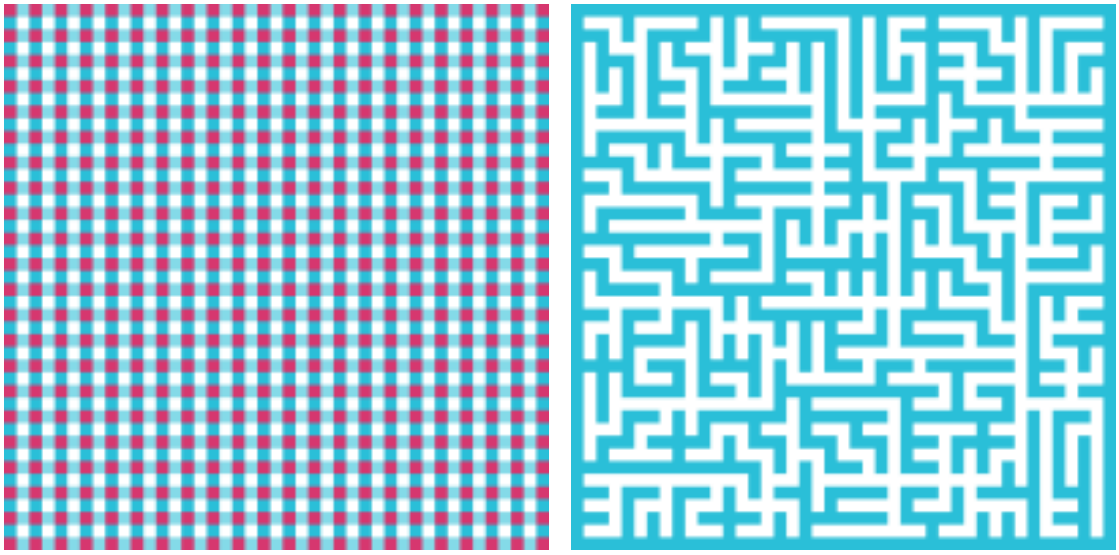
[5] `http://gamelogic.co.za/2013/12/18/what-are-grid-colorings/`

Algorithm 6 Randomized Prim's algorithm for grid colorings

```
                                        ▷ Initialization and generation operations
Apply grid colorings.
Add start cell room of the player, to the path;
Add the four walls adjacent to the room to the wallList;

while While the wallList is not empty do
    select randomly a wall from the wallList;
    Find the rooms adjacent to the wall;

    if Number of adjacent rooms = 2 and one of them not in path then
        Dig the wall: turn it into a room;
        Add the next unvisited room to path;
        Add the walls adjacent to the unvisited room to the wallList;
    end if

    Remove the wall from the wallList;
end while
                                        ▷ Post-processing operation
Remove randomly r * wallsCount walls;
```

Our choice falls on *vanilla Prim's*[6], where setting $u = (2,0)$ and $v = (0,2)$, and setting a $K = 3$ parameter, allows us to define a 2x2 square pattern. At (0,0) there is the first colour to a free cell; at (0,1) and (1,0) there is the second colour that is a wall cell, and at (1,1) there is the third colour that is a pillar. This colouring and cell assignment satisfies the three conditions stated before and so maze generation using Prim's algorithm for colouring grids is possible. Specifically in our case, this division always assigns a free room to the start position of the player and the other cells follow the vanilla Prim pattern.

[6]This correspond to the initialization of the grid colorings framework in order to obtain a maze according to the common notion of Prim's maze as discussed in 2.5.2

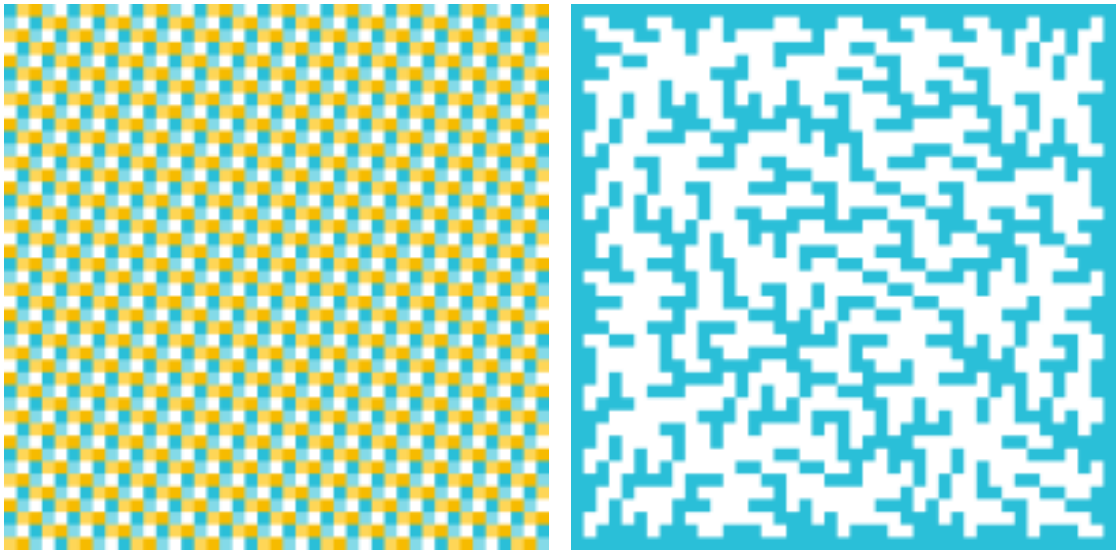


(a) Grid coloring initialization.

(b) Result after Prim's algorithm for grid colorings.

Figure 4.6: A 43x43 square grid initialized for vanilla Prim's and after the generation algorithm. Red cells are pillars, white cells are free rooms and light blue cells are walls[58].

The Prim's algorithm just introduced applied on vanilla Prim's colorings normally produces a perfect maze and, thanks to a further post-processing step, allows to generate also a non-perfect one.
Many different grid colorings and category cell assignment are possible, bringing to different possible results: some of them are biased to certain substructures and shapes.



(a) Grid coloring initialization.

(b) Result after Prim's algorithm for grid colorings.

Figure 4.7: A 43x43 square grid with $u = (5,0)$, $v = (3,1)$ where $k = 5$ colors are used. The two yellow cells are walls while the blue ones are pillars and white cells are free rooms[58].

Apart from the versatility and generality of the grid colorings abstraction, we will just consider vanilla Prim's in our thesis work. Some strange behaviour can happen due to the nature of the grid colorings technique. In vanilla Prim's, if the maze dimensions are both even numbers and if the end cell is on the opposite corner of the start, the end cell can become unreachable: this can happen also when the end cell is a pillar on the map border. Both of these problems can be easily solved enabling the autosolver. A last problem concerns the end cell when it is a pillar: without any post-processing operation the algorithm can produce a non perfect maze due to the pillar that must be a free room (since it is the end cell): this case is negligible since the end cell is not a passageway but the end of the challenge itself.

4.2 Alias Challenge Generator

As soon as the real map is decided in the previous component (Map Generator) we can access to the Alias Challenge Generator, that is responsible for the alias generation and analysis (through charts and statistics) of the problem discussed in

chapter 3. The Alias Challenge Generator is formed in turn by three components: the *Alias Map Generator*, the *Pace Chart* and the *Alias Challenge Optimizer*. The Alias Challenge Generator component can be accessed with the relative button "Go To Alias Generation" in the Map Generator but only when a real map present in it. Before accessing it some parameters must be set: we will here illustrate one part of them and the other part will be shown and illustrated later on.

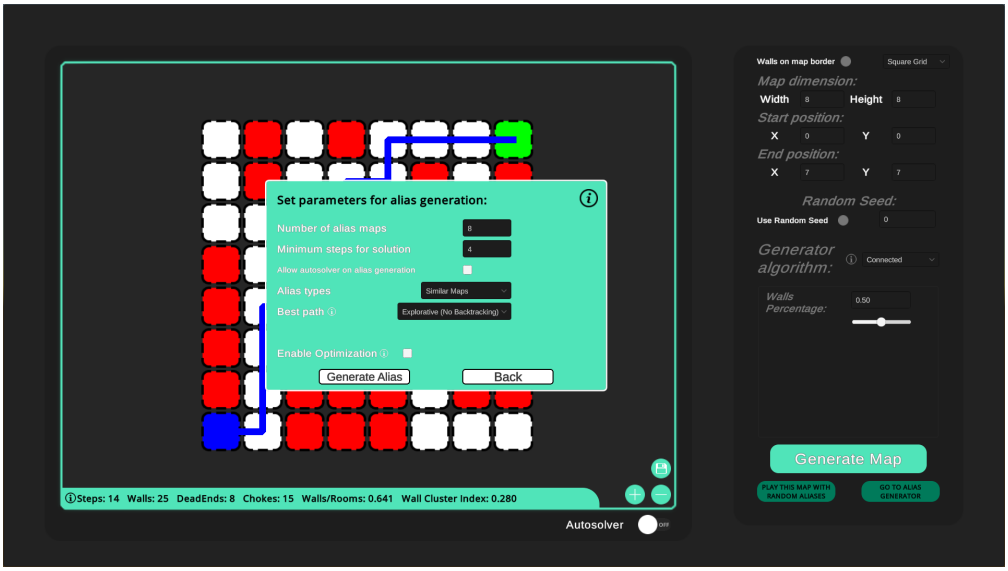


Figure 4.8: A screenshot of the parameters window (in light blue) before alias generation: when optimization is disabled only the alias generator parameters are shown.

- *Number of alias maps (k):* selected aliases that will be part of the set shown by the designer during the game.
- *Minimum steps for solution:* this parameter states that all the cells that the player can explore in the real map doing k steps, must be the same in all generated aliases from their respective start cell (that in general could differ from the real map). These cells are selected by doing a flood-fill on the real map from the start cell simulating all the possible cells where player can come up doing one step at a time. The cells that the player can find traversing the maze doing k steps are directly copied on all the generated alias maps: this copy allows that all the possible alias challenges that we can face by changing the alias set will surely last at least k steps.
- *Allow autosolver on alias generation:* this enables or disables the autosolver during alias generation in order to respectively make all the generated aliases valid or not (an alias is considered "valid" as discussed in 4.1.2).

- *Alias types:* the alias map generation process follows a quality metric that in our case dictates how much the alias map resembles the real map. With this *similarity metric* we decide if in the alias set we want similar, dissimilar or both type of maps. A thorough explanation on the metric choice will be done later in this section.

- *Best path:* for the game proposed, the "best path" wording refers to all the paths that allow to rule out the alias maps in the maps set. We developed two ways of finding the best paths, one fast but less precise (*Explorative*), and the other one reliable but very slow (*Explorative+Backtracking*): both of them use state-space search. The first one assumes that a player is usually interested only in exploring new cells and subsequently avoids walls that can give him/her penalties, while the latter is more "rational" in the sense that it also considers walls if some alias can be ruled out by doing that: this means that it can provide the designer with the real minimum and maximum number of steps that the helpers need to cooperate with the player in order to rule out the aliases. The search strategy adopted is the well-known *Breadth-first* search[56] since we want to get not just one, but all the best paths possible at each step made by the player (i.e. depth of the search tree).

  1. *Explorative:* Only new unexplored cells are visited by this search. The path does not backtrack[7] or loop[8]. Every time the search finds a wall cell, it stops because it would be forced to backtrack to continue the state-space search. This means that wall cells (that can only be visited) are considered only if by doing that the the real map is found. This behaviour allows to explore few states and so we obtain a relatively fast search time compared to "Explorative+Backtracking".

  2. *Explorative+Backtracking:* is the same as the explorative one, but with the possibility of backtracking. Walls (that can occur as dead ends sometimes) can be visited only if they allow to reduce or finish the aliases in the alias set. The problem is that the backtrack feature is extremely time consuming and requires a big state space (wrt to the one described in "Explorative").

When all parameters are set and the "Generate Alias" button is pressed, the Alias Map Generator acts to produce the set of alias map requested by the designer.

[7]proceeding by going back along the same path

[8]visiting a cell already covered by the path

4.2.1 Alias Map Generator

With regards to the classification proposed in 2.3, the Alias Map Generator is an offline, necessary, random seed, and also a stochastic and generate-and-test generator, with the goal of producing a list of alias maps. More precisely, the generation of a single alias map is done by taking a map with dimensions, start cell and end cell identical to the real map, and using the three generation map algorithms discussed in 4.1.3, 4.1.4 and 4.1.5 with the specific parameters and seeds completely randomized. This is done a number of times that should be enough to produce a vast portion of alias maps. Of the generate-and-test, the test part is simply the validity check for maps discussed in 4.1.2: an invalid map is discarded or "fixed" using the autosolver if enabled in the parameters shown before alias map generation.

The final generated content (list of alias maps) is finally according to a similarity metric, and the N (the "number of alias maps" discussed before) most similar or dissimilar aliases are shown to the designer: this can be chosen setting the "alias types" parameter. One can argue that using a similarity metric we are implicitly doing search-based PCG that uses a fitness function to evaluate the content (that in our case corresponds to the similarity metric). That is because we are not interested in a continuous search (i.e. traversing a huge number of states) to find an optimum but rather in finding local-optima in a reasonable amount of time: again, in this setting only the designer has the final word on what is "optimal" or not. As a CAD tool we want to at least allow the designer to test some dynamics that can happen when maps visually resembles or not the real one (also the "alias" word suggests in the first place a similarity with the real maps), since this choice could bias the helpers towards some alias choices during the game. The problem now is to find a good similarity metric that dictates the distance between the real map and a newly generated alias.

We want a similarity metric that can easily adapt to our issue, one that is easy to compute and can detect jittered version of the real map. Due to its versatility, low computational cost and resiliency from small perturbations (deformation)[59] we have chosen the *Mahalanobis distance* (extensively discussed in appendix A) that is reformulated as:

$$D^2_{\boldsymbol{G}}(\boldsymbol{I_x}, \boldsymbol{I_y}) = \sum_{i,j=1}^{d} g_{ij}(x_i - y_i)(x_j - y_j) = (\boldsymbol{x} - \boldsymbol{y})^T \boldsymbol{G}(\boldsymbol{x} - \boldsymbol{y}) \tag{4.1}$$

Where $\boldsymbol{I_x}$ and $\boldsymbol{I_y}$ are two given samples/instances (in our case the mazes) for which we want to compute the distance D. Given a generic coordinate k in the maze, x_k and y_k are the values (in our case 1 if there is a wall, 0 otherwise) of a cell of the maze at position k. Our function choice for G is a Gaussian function so that the

conditions for a valid pseudo-metric are directly entailed[59].
From the Gaussian function we want that the value halves from the maximum at $|\boldsymbol{C}_i - \boldsymbol{C}_j| = 1$ (orthogonal cell) and that at $|\boldsymbol{C}_i - \boldsymbol{C}_j| = \sqrt{2}$ (direct diagonal neighbour cell) the value of the Gaussian is less than or equal to $\frac{1}{4}$ the maximum value of the Gaussian[9]. A Gaussian function that satisfies all our design constraints is $\mathcal{N}(0, \frac{3}{4})$:

$$g_{ij} = f(|\boldsymbol{P}_i - \boldsymbol{P}_j|) = \frac{1}{\sqrt{2\pi\frac{3}{4}}} e^{-\frac{|\boldsymbol{P}_i - \boldsymbol{P}_j|^2}{2\frac{3}{4}}} \quad where \quad \boldsymbol{G} = (g_{ij})_{dxd} \tag{4.2}$$

and finally our distance metric squared:

$$D^2_{\mathcal{N}(0,\frac{3}{4})}(\boldsymbol{I_x}, \boldsymbol{I_y}) = \frac{1}{\sqrt{2\pi\frac{3}{4}}} \sum_{i,j=1}^{MN} e^{-\frac{|\boldsymbol{P}_i - \boldsymbol{P}_j|^2}{2\frac{3}{4}}} (x_i - y_i)(x_j - y_j) \tag{4.3}$$

Since g_{ij} is a Gaussian function, all the conditions for the distance metric are satisfied (see the work done by Wang et al.[59, 60] and the appendixA for all the details).

Given our formal definition, some implementation changes where taken to improve computation and give a better understanding of the distance $D_{\mathcal{N}(0,\frac{3}{4})}$ metric to the designers.

- We avoid the computation of matrix G, especially when the map dimension changes. This is implicitly done by directly assigning the right g_{ij} value in the dot product of the distance in equation 4.1.

- To have a better understanding of the "distance" between two maps, we increase the scale of g_{ij} by a factor that makes its maximum value equal to one.

- Values of $g_{ij} \leq 0.1$ (i.e. given our definition , all the cells are outside Moore neighbourhood of range one) are considered irrelevant for single cell similarity and so counted as zero.

[9]The quantity $|\boldsymbol{C}_i - \boldsymbol{C}_j|$ computes the euclidean distance between cell coordinates at i and j positions.

4.2.2 Pace Chart

This component was inspired by the work of Sorenson et al.[61, 62, 63] on automatic generation of "fun" levels. We will do a brief introduction on it before moving on to the work done.

From its very beginning, the problem is quite complex, since the definition of "fun" in the videogame context is broad and debatable. According to the authors, the relation between "challenge" and "fun" seems very strong in the literature and so they try to relate those two concepts while maximizing the fun for the player on the generated content[10]. They chose a periodic model introduced by Smith et al.[64] based on *rhythm groups*: non-overlapping sets of the game level sections that encapsulate a challenging moment of gameplay. The videogames inspected are 2D platforms and 2D top-down adventure games where the player has to go from start to a goal position doing some actions (e.g. killing enemies, climbing, jumping and so on). They find rhythm groups along the path of the player from where he/she starts, up to the end goal. In order to do this, a game-specific definition of challenge $c(t)$ at time t is required: this function must encapsulate perceived difficulty a player experiences at time t. For *Super Mario Bros.*[11] the challenge during a jump between two platform is measured by making it directly proportional to the number of potential trajectories that successfully traverse the gap; while for a game like *The Legend of Zelda*[12] the challenge during dungeon traversing (along the shortest path) is the number of enemies inside the the room at instant t.

[10]This does not exclude that other relevant components exist (and this is actually the case), but the authors have inspected it only regarding the fun-challenge relation.

[11]Nintendo EAD. "Super Mario Bros.". Nintendo, 1985

[12]Nintendo EAD. "The Legend of Zelda". Nintendo, 1986

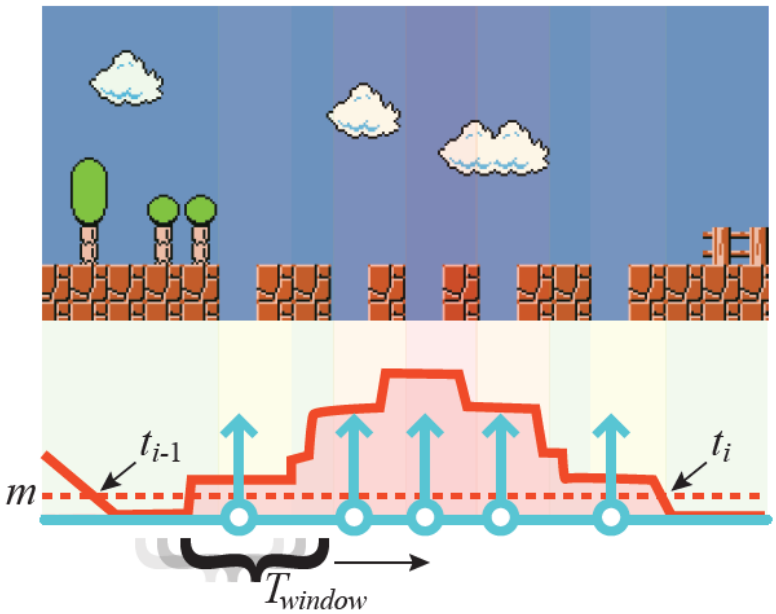


Figure 4.9: Illustration of rhythm group i in the context of a *Mario* game level. Vertical arrows represent the $c(t)$ that, accordingly to the authors, are Dirac delta functions (impulses) placed along the game level. Rhythm groups are identified by containing a relevant amount of challenge (above a constant value m): this rhythm group is located between t_{i-1} and t_i. Since in *Mario* going through a plain platform has $c(t) = 0$, a level section that contains platform jumps is more reasonable to consider in the fun computation. The accumulated challenge in the entire rhythm group (c_i) corresponds to the integration of the impulses located between boundaries t_{i-1} and t_i

Sorenson et al. have given different and more refined definition of the fun function over time[61, 62, 63]. We are not interested in expliciting precisely this function rather than listing the features and trends that the model introduced by the authors wants to convey with this rhythm group-based approach, that are directly derived from the concept of *Flow*[13]. Given a specific rhythm group the following holds:

- When the rhythm group starts, the fun increases according to the cumulative challenge up to the considered instant.

[13]Concept introduced by *Csikszentmihalyi* in *"Flow: The Psychology of Optimal Experience."* and refined in [65] that inspired the authors.

- When the accumulated challenge is above some threshold M, the fun starts to decrease: excessive challenging levels do not cause fun to the player.
- On the contrary, low level challenge (below m) are not considered so much fun for the player and so the f does not accumulate below that threshold.

The two thresholds m and M can make the difference in building a good challenge for the player, since they dictate the minimum and the maximum of difficulty for a level in order to match the player's skills and make the level entertaining and fun to play for him/her. Unfortunately finding m and M is really difficult and game dependent, but the framework proposed by Sorenson et al. is of big help in explaining the rationale behind the decision of computing interesting paths and analyzing their pace during the development of this thesis work.

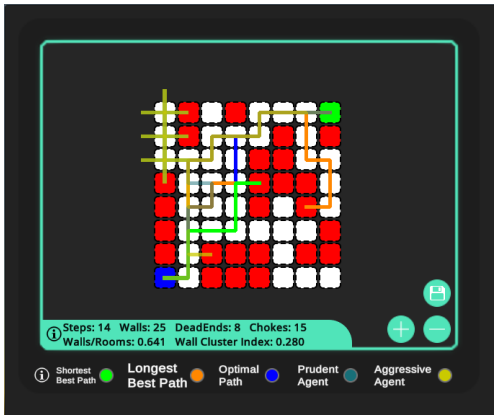
CMP is able to show to the designer five different and interesting paths on the real map with their respective pace in the Alias Map Generator.

1. *Shortest Best Path:* this is the absolutely best path (discussed at the beginning of 4.2), with the lowest number of steps[14].
2. *Longest Best Path:* is the best path (discussed at the beginning of 4.2) and with the highest number of steps.
3. *Optimal Path:* is the path computed using the well-known A* algorithm.
4. *Prudent Agent Path:* is the path computed by simulating a maze-traversing agent that tries to reach the end considering the most probable next cell (so implicitly considering as "true" H alias maps) that does not produces a penalty (i.e. room cells or border). Ties on the next cell choice are broken by taking the cell that reduces the distance with the most promising end cell location (computed averaging the end cell location among all the end cells H).
5. *Aggressive Agent Path:* is the path computed by simulating a maze-traversing agent that tries to reach the end considering the less probable cell as the next cell (so implicitly considering as "true" K alias maps): if the guess turns out to be true, the agent will be able to rule out a lot of maps sooner. Ties on the next cell choice are broken by taking the cell that reduces the distance with the most promising end cell location (computed averaging the end cell location among all the end cells K).

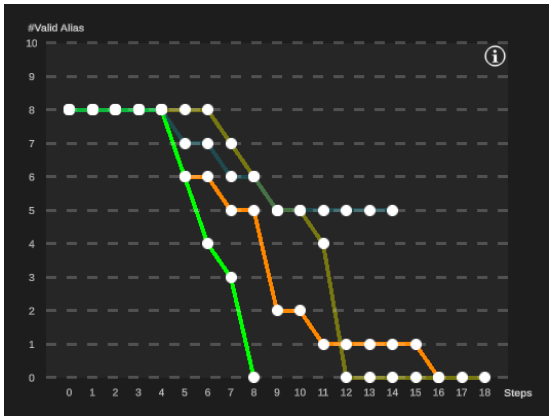
[14]We have already given a definition of "best path" at the beginning of 4.2 and now the same term reported here can be misleading. When we will talk about an actual "path" we will implicitly intend the definition given here while when we talk about the computation of paths, we intend the algorithm or the process to find all the paths that rules out all the aliases in the alias set.

One could argue the usefulness of having two agents that do not model precisely the player. In his paper Nelson[66] proposes that game metrics (information collected from players in playtests) are not the only source of metrics: not all the information needs to (or ought to) come from empirical playtests, since the game itself can say a lot to the designers. One of the strategies that he propose to retrieve this information is *hypothetical player-testing*: we assume a particular or simplified player model (i.e. the agents). Our chosen models are inspired partly by the character classes common in many games: one prudent agent that prioritizes its own safety instead of rushing to the end goal and an aggressive one that, conversely, wants to finish the challenge as soon as possible. One can also argue that our are character stereotypes choices, but our will with this two agents is to allow the designers "to investigate how the game operates in various extreme or idealized cases"[66].

For each of these paths we also compute the number of valid alias at each step. This information is the one we show in the chart. Given the framework of Sorenson et al. discussed at the beginning of this part, we can make similar reasoning for the videogame proposed in this thesis. Surely, the rhythm group is one and starts at the beginning of the challenge up to the end of it (all alias ruled out). The challenge $c(t)$ is proportional to the number of alias maps and is a delta Dirac (impulse) at time zero: it decreases when, going along one chosen path, we can rule out one or more alias maps (one or more negative impulses). Ruling out all aliases means that the challenge is finished, and $c(t)$ is zero from that point and on. Since we are not in the role of deciding how much a given alias challenge is complex (m, M and the $c(t)$ for a given alias map) we are not able to set a precise $c(t)$. We can at least assume that this proportion is equal one, that the integration interval (T_{window}) is equal to one and that if we have N maps the Dirac at $c(0) = N$.



(a) Real map in the Alias Challenge Generator.



(b) Pace Chart in the Alias Challenge Generator.

Figure 4.10: Those two parts are strictly related in the Alias Challenge Generator. Clicking the name on the path names allows to show or hide the path selected. The decreasing of the number of alias is reported in the Pace Chart on the right.

Viewing the decreasing of the challenge in one path on one or more steps is a valid tool for a *what-if* analysis and for further optimizations of designer's interest. This chart has curves that reflect a quantity directly relative to the challenge and so to the level of fun that the game intended to reach.

4.2.3 Alias Challenge Optimizer

This component allows the optimization of the functions shown on the Pace Chart component. We will firstly show the parameters and the type of optimization that the designer can choose through the user interface (UI), and lastly we will talk of the optimization algorithm chosen.

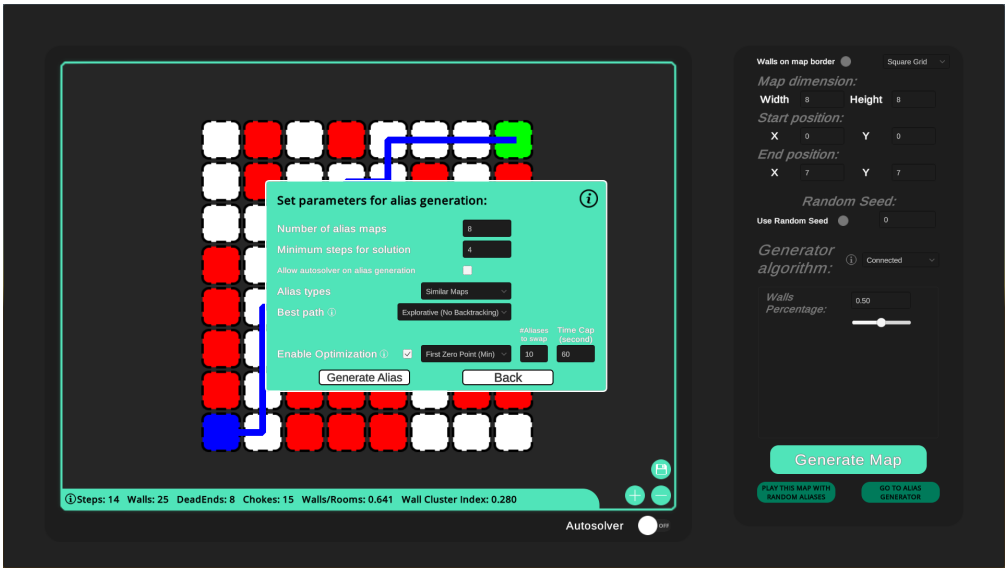


Figure 4.11: A screenshot of the parameters window (in light blue) before alias generation: optimization is enabled so the relative parameters are also shown.

- *Optimization type:* the quantity to optimize (minimize or maximize) selectable through the drop-down menu (in the above screenshot "First Zero Point (Min)" is selected).
- *Number of Alias to swap (M):* the number of alias maps to generate and try to swap in the alias set at each iteration of the optimization algorithm.
- *Time cap:* After the time expressed by this parameter (in seconds) elapsed, the optimization algorithm terminates giving back the best alias set it founds in the given time-span.

The optimization algorithm used is the well-known *Hill-Climber* that comes directly from *hill climbing* as a mathematical optimization technique. This algorithm initially generates a random state and iteratively moves towards neighbours states until the function to optimize stop to increase. Occasionally the algorithm returns the global optimum (and also in that case we cannot be 100% sure that is a global one) but rather a local optima, depending on the state initialization. More precisely, we will use the *simple hill-climbing* where each time we do not generate all the neighbours but a fixed number of them and move to the best one found. As already said, with this project we are not interested in finding optimal solutions since local optimal are a still a valid solution in the CAD framework and the time spent in optimization should not be extensive. With the Hill-Climber algorithm we are able to offer to the designer a simple tool to investigate local optima solutions

that still represent peculiar cases that can be viewed, analyzed and tested by the designers.

Algorithm 7 Simple Hill-Climber for Alias Challenge

```
        ▷ generateAliasSet(N) function randomly generates an alias set of N
  aliases using the algorithms of the Map Generator component
  currentNode = generateAliasSet(N);

  for ever do
      aliasMapsToSwap = generateAliasSet(K);
      Initialize nextEval integer to the lowest value possible;
      Initialize nextNode;
      Assign to currEval the result of the function f to maximize computed on
  currentNode;

      for all possible swaps between one map in currentNode and one map in
  aliasMapsToSwap, create swapNode do

          Assign to swapEval the result of the function f to maximize computed
  on swapNode;
          if swapEval > nextEval then
              nextNode = swapNode;
              nextEval = swapEval;
          end if
      end for

      if nextEval ≤ currEval then
          returncurrentNode;
      end if

      currentNode = nextNode;
  end for
```

The Hill-Climber algorithm by definition tries to maximize the value of the function f. Minimization is possible by inverting the sign of every variable that contains the value to optimize (i.e. the ones that ends with *Eval* in the pseudocode). Lastly, we explicit all the function that the user can maximize or minimize, that, evaluates metrics of the alias challenge (real map plus alias set) mainly inspecting the curves displayed on the Pace Chart4.2.2:

- *First Zero Point (Min):* minimizes the first point that goes to zero on the

Pace Chart considering all the displayed lines.

- *Best Path Pitchfork (Max):* maximizes the first and the last point that go to zero in the Pace Chart considering only the Best Path curves.

- *Agents Pitchfork (Max):* maximizes the first and the last point that go to zero in the Pace Chart considering only the agents paths curves.

- *Overall Pitchfork (Max):* maximizes the first and the last point that go to zero in the Pace Chart considering all the displayed lines.

- *Reliability Best Path (Min):* minimizes how much is likely that the path of the player will incur in one of the Shortest Best Paths that could be displayed on the map (simply by removing and re-inserting the same map in the alias set).

- *Reliability Best Path (Max):* maximizes how much is likely that the path of the player will incur in one of the Shortest Best Paths that could be displayed on the map (simply by removing and re-inserting the same map in the alias set).

4.3 Game Level

From the button displayed near the bottom-right corner of the UI we can play the actual game level in both the previous components (Map Generator and Alias Challenge Generator). A real map is required for the challenge to be tested. When in Map Generator, the play button will allow the designer to test the alias challenge randomizing the alias set. When in the alias challenge generator, the play button effectively proposes the alias challenge displayed on the UI with the combination of real and alias set defined at that point. Before testing the alias challenge (i.e. play the prototype of the game proposed) the designer can set some parameters for the challenge:

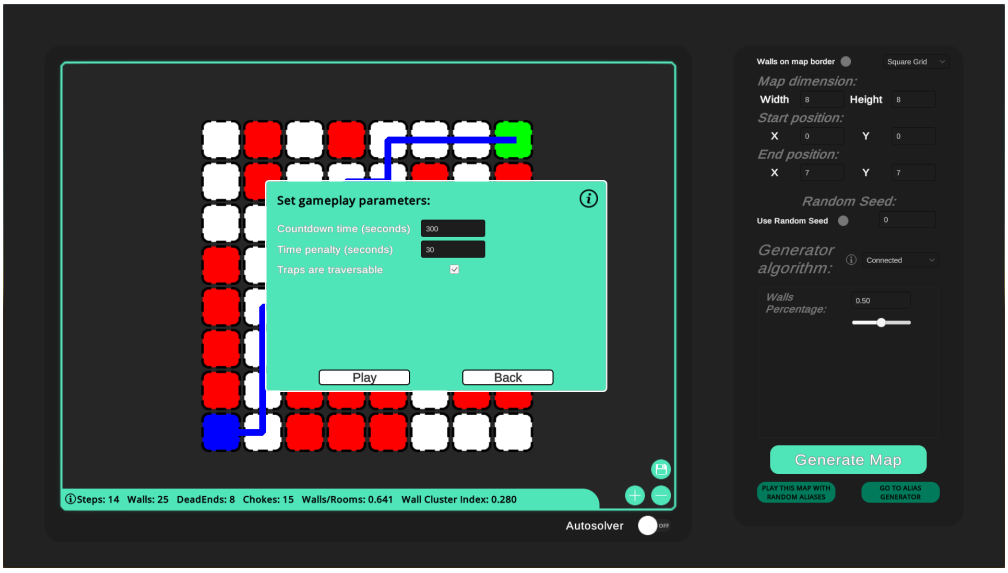


Figure 4.12: A screenshot of the parameters window (in light blue) before playing the alias challenge: in this case the button was pressed in the Map Generator component.

- *Countdown time:* is the timer that will be set for the alias challenge. When it reaches zero the challenge ends with a game over.
- *Time Penalty:* it is the time penalty that the player gets when accessing a wall cell during the gameplay.
- *Traps are traversable:* when this toggle is on, the player can traverse wall cells during the gameplay.

Chapter 5

Experiments on optimization

In this chapter we will firstly state some adjustments done in order to set up a satisfactory and usable setting for optimization, and after that we will extensively show the experiments done, while explaining the intentions and conclusions supported by explanatory graphs and plots.

5.1 Optimization refinements

5.1.1 On optimization algorithm

In algorithm 7 we have introduced our optimization algorithm that coincides with a simple hill-climbing algorithm. This choice, again, is due to the fact that the number of neighbour nodes/states is extremely high, both for the variety of possible aliases that can be generated, and also for all the possible permutations between the aliases contained in the initialized alias set and the number of aliases to swap (M aliases 4.2.3). This disallows the generation of all possible neighbours in our version of hill-climber that converges on few iterations of the outermost for loop. To get the best from the optimization phase, we have decided to build on top of the algorithm 7 the *Random-restart hill climbing* (R-R): this algorithm simply does an hill-climber K times, finally choosing the best node/state (alias challenge in our case) found (i.e. that has the maximum value of the function to optimize). To

get the "best" instance found, the well-known *priority queue*[1] abstract data type is used. In the context of our work, the random-restart algorithm becomes:

Algorithm 8 Random-restart Hill-Climber for Alias Challenge

▷ $simpleHC(N)$ is the algorithm 7 that finds an N alias set.
▷ $EvalF$: result of the function f to optimize, evaluated on the alias challenge.

for K times **do**
  $AliasSet = simpleHC(N)$;
  $priorityQueue.Enqueue(AliasSet, EvalF(RealMap, AliasSet))$;
end for
$return \quad priorityQueue.Dequeue()$;

5.1.2 On time spent in optimization

As far as the time required to execute the algorithm is concerned, we have found that the bottleneck shows when the function to evaluate requires the computation of the best paths: five of the six functions (agents pitchfork excluded) require the computation of the best paths. This happens because two the state-space search algorithms for finding best paths (described at the beginning of 4.2) continue independently from the height of the tree. Pruning the search up to the minimum length of the best paths (the Shortest Best Pats as explained at the end of 4.2.2) results in a huge speedup in the computation of the best paths, with the only drawback of not considering longer ones. This choice is reasonable since Longest Best Paths are very unlikely to be taken by the real player inside the game level and so can be considered uninformative (especially if they are few). This pruning technique is added on top of the most computationally expensive (but also the more precise wrt player behaviour) state-space search available: "Explorative+Backtracking". This new version is called "Explorative+Backtracking (fast)", where the first version will be considered the "slow" one.

[1]Every element of the queue has an associated priority: *enqueue* opertaion inserts an element with a given priority in the data structure, while *dequeue* takes the element with the highest priority from the data structure.

Best Path search algorithm	Average computation time (s)
Explorative	0.084
Explorative+Backtracking (fast)	0.021
Explorative+Backtracking (slow)	23.280

Table 5.1: Average time of the best path computation in different state-space search settings for a given fixed alias challenge.

As we can see from 5.1 table, we have a huge speedup with the fast version of the same algorithm, so that it surpasses the fast and less precise "Explorative" algorithm. The "slow" version is one thousand times slower than the first two: time complexity is exponential wrt the maximum depth reached by our search, which, most of the times, is shallower in the "fast" version of algorithm than in the "slow" one. In the experiments that will be introduced in the next section, all we needed was an algorithm that retrieves the best paths precisely and with a low computation time. The "Explorative+Backtracking (slow)" is the ideal solution for carrying out a lot of experiments. Unfortunately, we still lose the contribution of the Longest Best Paths, but in our function optimization trials those paths are needed only once and, even in that case, the quantity optimized is still valid and useful for the designer.

5.2 Optimization function validation

In this part we want to evaluate how "good" are our choices on the optimization functions discussed in 4.2.3. The validation process basically tries to observe if, given our chosen algorithm 8, the best instance found is better than the one found doing random search(R-S) (obtained by simply randomizing all the parameters in the three previously discussed generator in 4.1.3, 4.1.4 and 4.1.5), with the same number of trials/iterations for both algorithms. The number of "trials" for the R-R are the total swaps done in the simple hill-climber that the algorithm uses, while for the R-S this number simply corresponds to the number of alias set randomly generated.

In order to do this, we have started from a data collection, i.e. a big run of algorithm 8 to observe the frequency of the optimal value. This allowed us to set up the right K for the R-R algorithm in order to retrieve the best value with high probability in all the algorithm runs. After the K is found, we have done our experiments of collecting the values in a *comma-separated values* (CSV) file. Every

single experiment will produce a CSV file, where each row contains the value of the function to be optimized, evaluated on a given swap computed by algorithm 7 (simple hill-climber); and in the other column we will find the k-th iteration of the caller (R-R) algorithm, where the aforementioned value for the swap has been computed.

For all the functions considered, the number of experiments performed was $E = 10$. The real map chosen is an 8x8, where the start cell is located at (0,0) and end cell at (7,7). The map is generated by the connected generator with the random seed fixed at zero and the initial wall percentage set to one-half. The alias generation and challenge optimizer parameters were set as shown in the following image:

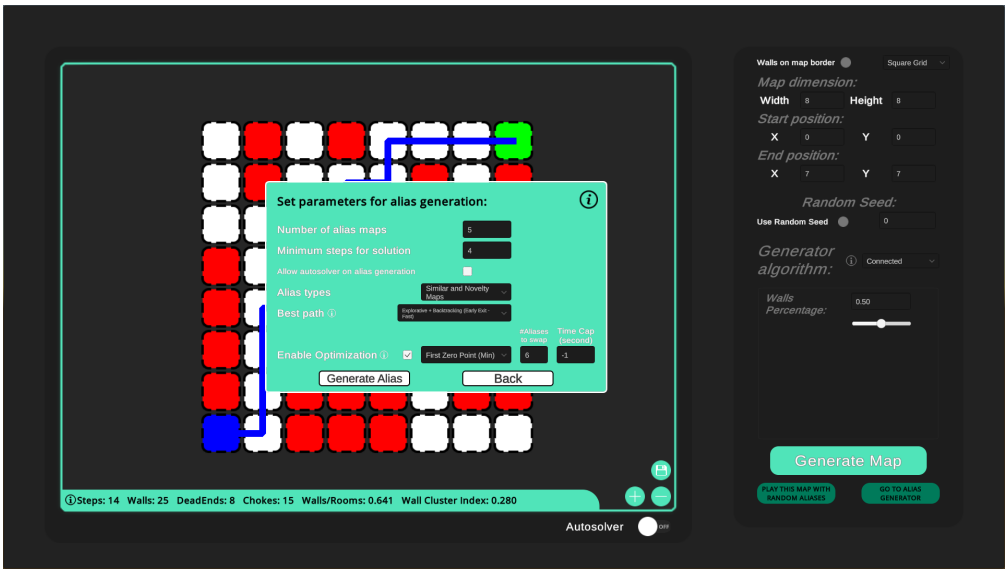


Figure 5.1: A screenshot of the parameters window (in teal) before the experiment runs. The number of K has been embedded in code and changed in every experiment (if necessary). Motivations for best path search algorithm choice have been stated in 5.2.2. Negative time cap will translate in no time constraint, so the experiment can produce all the values without problems. Number of aliases equal to 5 is an average number of aliases according to the author, while the number of 6 aliases to swap is a good choice, since it is a small set of maps that are generated with a balanced variety, if that number is a multiple of three[2]. A total of $5 * 6 = 30$ number of swaps are tried every time the outermost loop of the simple hill-climber algorithm runs: this is a sufficiently high number of swaps that still allows a reasonable execution time for experiments.

In the next pages the optimization function validation is discussed one function at a time, without considering the overall pitchfork: in fact, considering it could be really misleading in our setting, given our choice of the Explorative+Backtracking (fast).

5.2.1 First zero value validation

With this optimization we want to minimize the first zero value, that will be the one of the Shortest Best Path (green line in the Pace Chart) in practically all the cases. So with this optimization function, we want to obtain at least one of the Shortest Best Paths.
Before showing the result, a little note must be done. As we can see from figure 5.1, "Minimum steps for solution" is equal to 4 in all our experiments instances. This will mean that all the cells that the player can visit in 4 steps will be copied from the real map and pasted to every alias that we generate. This implies that we cannot go equal or under 4 with the first zero value function, and so that the optimal minimum will be precisely 5 in our experiments setting.

[2]The choice of which generator to use follows a *round-robin* fashion: since the available generators are three, it is preferable to generate a batch of aliases to swap which is a multiple of three, so that their production of aliases is balanced.

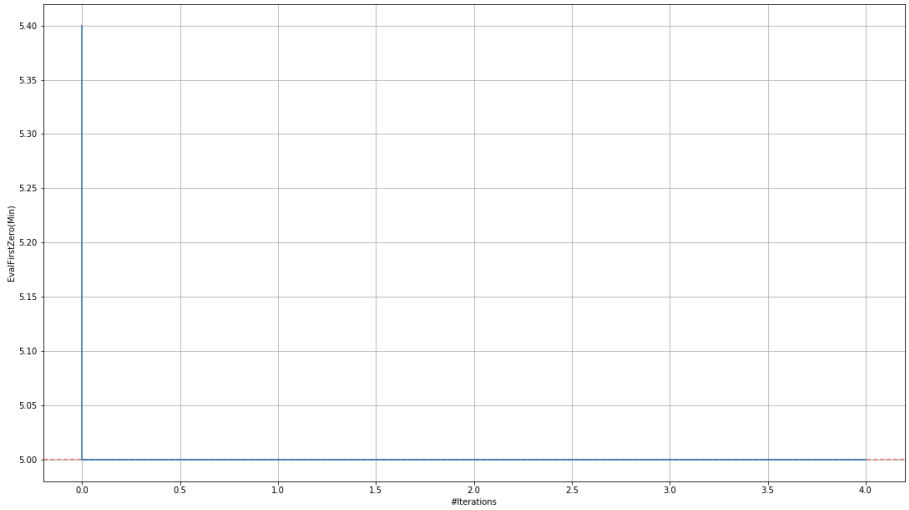


Figure 5.2: The mean optimal values (averaged on $E = 10$ experiments) found by random-restart hill-climbing algorithm in $K = 5$ iterations. The red-dotted line shows the best value found by random search.

The first thing that we can notice is that this function to optimize converges to the optimum very soon, and this can be noted by the choice of K that (as stated before) is big enough to guarantee that the optimum is found with high probability: we can see that, in all the experiments, the optimum is found in the second iteration of the random-restart. Besides, in our validation tests, one iteration of random search is sufficient to find the optimal value and it is less than the average number of swaps in our version of random restart.

In this case it's evident that we do not take the best from the optimization algorithm since it gives the same result in more iterations compared to the random search that converges in one iteration (according to our tests). Reasonably, if the number of aliases in the alias set increases, this function could become more difficult to minimize and we could potentially obtain opposite results.

5.2.2 Best path pitchfork validation

This function evaluates and maximize the difference between the zero of the Shortest Best Path and zero of the Longest Best Path found. The problem is that in the best

paths search defined in , Longest Best Paths are not computed and are assigned as zero vectors in the code. This means that this optimization actually maximizes the first zero value of the Shortest Best Path (green line in the Pace Chart) giving to the designer the possibility to maximize it and see how much further we can bring the minimum steps required to solve the alias challenge.

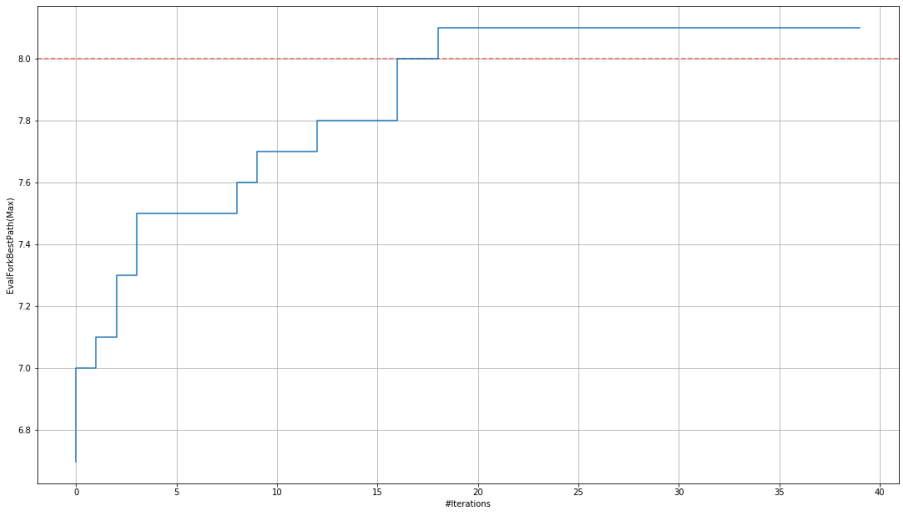


Figure 5.3: The mean optimal values (averaged on $E = 10$ experiments) found by random-restart hill-climbing algorithm in $K = 40$ iterations. The red-dotted line shows the best value found by random search.

As said in 5.2.1, the minimum for the first zero value is 5 and we maximize in all experiments up to 8, except one in which we can reach 9. We reach a value better than the one we would reach with the random search, but again, it is not worth to go for optimization in this case since our validation tests have proved that the same result can be obtained by a smaller number of iterations using random-search, and the 9 obtained in one of the $E = 10$ experiments is really unlikely to happen.

Again, we can not obtain the best when optimizing this function: surely different initial setting are needed to state this, but as for cases similar to this one, random search gives same results in less iterations wrt the optimization algorithm (R-R).

5.2.3 Agents pitchfork validation

This optimization function tries to maximize the difference between the first zero value of the aggressive agent and the prudent one (evaluating the respective curves on the Pace Chart).

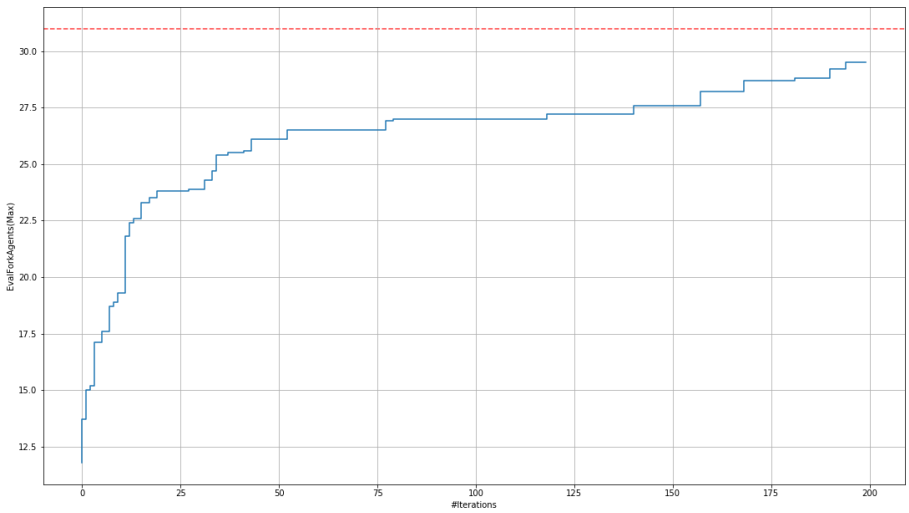


Figure 5.4: The mean optimal values (averaged on $E = 10$ experiments) found by random-restart hill-climbing algorithm in $K = 200$ iterations. The red-dotted line shows the best value found by random search.

The result of our validation tests is that not only the random search is better, but also that this quantity is too sparse and random in general. In our experiments the *interquartile range (IQR)*[3] of the number of swaps before finding the optimum during the $E = 10$ experiments on R-R is really wide: in addition, the values found by R-S with same number of iterations as swaps gives better results on average. We argue that this function, when optimized, is really useful: firstly because there is no control on which of the two agents should be the first; secondly, because the agent path in some cases is not only one (ties propagate until the next node decision) and this can be misleading for the designer.

[3]The difference between 75th and 25th percentiles: measure of statistical dispersion.

5.2.4 Reliability best path (min) validation

This function computes how many paths of length equal to the Shortest Best Paths are actually best paths: this fraction is then minimized.

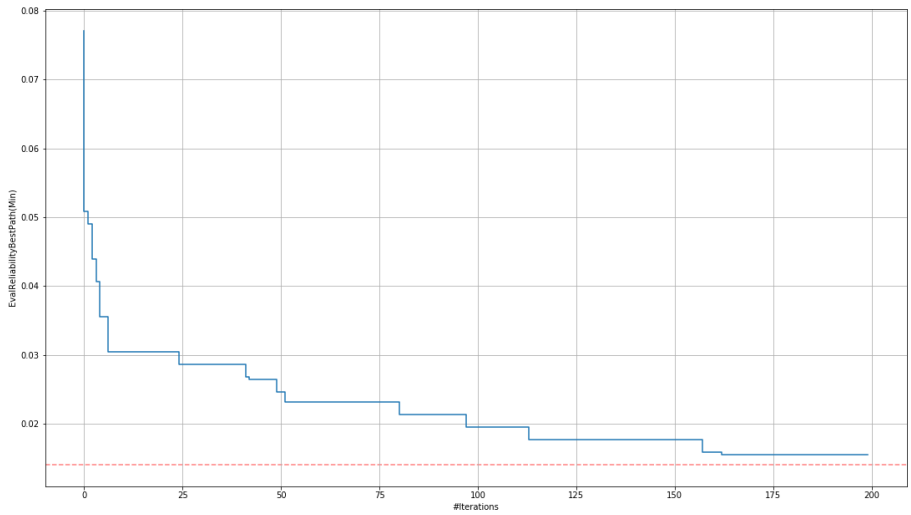


Figure 5.5: The mean optimal values (averaged on $E = 10$ experiments) found by random-restart hill-climbing algorithm in $K = 200$ iterations. The red-dotted line shows the best value found by random search.

In our tests we have found that the random search finds a better value wrt the optimization, but only after a lot of iterations and very just few times (more or less the double of the average of R-R swaps required to find the optimum). In addition, the difference between the mean optimal value of R-R and the one found in random search, is caused by the only one outlier (that does not allow the mean curve 5.5 to reach the value on the red-dotted line) where the other ones always find the same optimum value of random search. Apart from this outlier, the optimization algorithm is able to find an optimal value in less than half "trials" on average. We can so conclude that minimizing the reliability best path function is a valid and viable optimization process.

5.2.5 Reliability best path (max) validation

This function computes how many paths of length equal to the Shortest Best Paths are actually best paths: this fraction is then maximized.

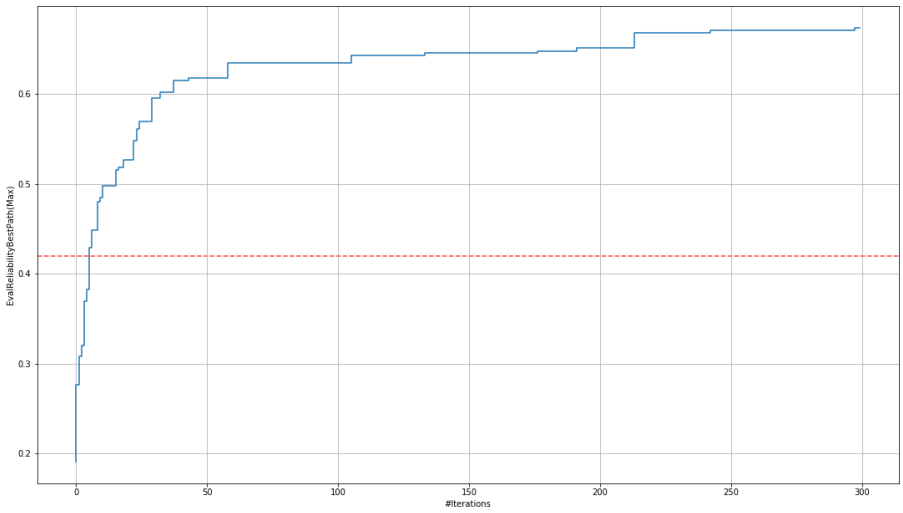


Figure 5.6: The mean optimal values (averaged on $E = 10$ experiments) found by random-restart hill-climbing algorithm in $K = 300$ iterations. The red-dotted line shows the best value found by random search.

From the figure 5.6 we can easily state the effectiveness of the R-R optimization algorithm for this function. We have always found better values than with random search (in less number of "trials" on average in our tests). This is probably due to the fact that this kind of function to optimize has an inner nature of finding a solution "step by step" adding one better element at a time: this behaviour is inherently included in our optimization algorithm (random restart hill-climbing algorithm).

Chapter 6

Conclusions and Future Developments

We have presented our work in collaboration with the startup *AnotheReality* to realize a tool meant to help the designers of the company to generate and evaluate particular game content for the VR cooperative experience they have proposed. With this work we have succeeded in realizing a tool that satisfies the CAD hypothesis. Its main components allow the designers to generate a diverse number of maps using PCG (the most used techniques for 2D level generation) and to operate a "what-if" analysis, by trying continuously different alias challenges through a intuitive and simple drag-and-drop system UI. The generated aliases proposed try to resemble (similar alias) or not (novelty alias) the one true map, and this similarity is measured according to a metric that fits well with our problem and it is easy to compute. In addition to this we have increased the degree of the designer-computer cooperation, giving the possibility to optimize some interesting functions. All of them can be potentially of interest for the designer, since they directly influence the shape and interpretation of the Pace Chart that proved to be a useful tool in measuring the "challenge" and consequently the "enjoyment" of the alias challenge proposed. The chosen functions were then validated, and while of them where of use, others need additional experiments with different aliases challenge instances provide some useful results. Actually, a last note could be done in term of the completion of the CAD system. Collaborative Maze Project succeeded in paving the way to a supporting tool for the videogame proposed: up to this point, the designer can test different cases and multiple gameplay scenarios. Apart from this, CMP is far from being a complete tool, but more feedback by experts (i.e. designers) is needed to refine and polish it.

Future developments can be the followings:

1. *Addition of waypoints:* those are simply 3D object in the game world that ease the player's experience in ruling out aliases in alias set. These waypoints are displayed as 2D token also on the map set available to the helpers. This new mechanic leads the way to different interesting problems, such as how many waypoints to have and where to spawn them in the game level and on the map set available to the helpers.

2. *Application to display the map set:* our game prototype fuses the role of the player with the role of the helpers, by giving the player the possibility to see the maps set. This works fine for a prototype, but to best test and catch all the positive or negative aspects of the videogame, it would be better to induce the cooperative behaviour giving helpers the possibility of using a different device (laptop or smartphone), where the maps set of the maze challenges is displayed to simulate a game scenario true to the one described by the company.

3. *More on experiments:* our analysis in chapter 5 was pretty useful and valid to assess our choice on optimization function. Since we lack a continuous designer's feedback, we can continue by analyzing how some function increase wrt to other map parameters. Catching this dependencies can be a big contribution in a CAD framework, since we can add or refine components to make it behave better on designer's input directly (e.g. disallowing some parameter configurations) or indirectly (e.g. showing some tips in the info boxes).

Appendix A

The Mahalanobis distance metric

Introduced by the homonymous statistician[67], this metric has been used in many scientific fields, such as computer vision and machine learning, and it includes and expands the common and usual *Euclidean distance* metric. The main usage for this kind of distance is to obtain a metric that quantifies how much a given test point $\boldsymbol{x} = (x_1, x_2, ..., x_d) \in \mathbb{R}^d$ is close to an observation one with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$.

$$D^2_{\Sigma^{-1}}(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu}) \tag{A.1}$$

Informally, the Mahalanobis distance computes the usual Euclidean distance (i.e. a number) of a target point when the observation are re-scaled into a normal standard distribution. Given a set of observations, this distance allows to classify the test point to closest one.

It can also be defined as a dissimilarity measure between two random instances $\boldsymbol{x}$ and $\boldsymbol{x}'$ from the same distribution.

$$D^2_{\Sigma^{-1}}(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x} - \boldsymbol{x}')^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{x}') \tag{A.2}$$

As done by[68, 59, 60] we will "consider that a *Mahalanobis distance metric* is any dissimilarity function parameterized by a *symmetric positive semidefinite* matrix $\boldsymbol{G}$". It is written in this form:

$$D^2_{\boldsymbol{G}}(\boldsymbol{I_x}, \boldsymbol{I_y}) = (\boldsymbol{x} - \boldsymbol{y})^T \boldsymbol{G} (\boldsymbol{x} - \boldsymbol{y}) \tag{A.3}$$

A given matrix $\boldsymbol{A} \in \mathbb{R}^{dxd}$ is symmetric if and only if it:

$$\boldsymbol{A} = \boldsymbol{A}^T \iff a_{ij} = a_{ji} \quad \forall i, j \tag{A.4}$$

A given matrix $\boldsymbol{A} \in \mathbb{R}^{dxd}$ is positive semi-definite (PSD) if and only if:

$$\forall \boldsymbol{x} \in \mathbb{R}^d, \quad \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} \geq 0 \tag{A.5}$$

Ensuring that $\boldsymbol{G}$ is a symmetric PSD matrix, defines a well-posed distance metric also called *pseudo-metric* (i.e. symmetric, non-negative values and the triangle inequality satisfied[1] by $D^2_{\boldsymbol{G}}$). Taking G as the identity matrix, defines the base-line euclidean distance metric.
A different way to design our PSD matrix $\boldsymbol{G} = (g_{ij})_{dxd}$ is to rewrite it as the result of the dot product:

$$D^2_{\boldsymbol{G}}(\boldsymbol{I_x}, \boldsymbol{I_y}) = \sum_{i,j=1}^{d} g_{ij}(x_i - y_i)(x_j - y_j) = (\boldsymbol{x} - \boldsymbol{y})^T \boldsymbol{G}(\boldsymbol{x} - \boldsymbol{y}) \tag{A.6}$$

In the framework of Wang et al.[59, 60], samples $\boldsymbol{I_x}$ and $\boldsymbol{I_y}$ are images represented as binary vectors where every term refers to a specific pixel binary value of the given image lattice sample: pixels can be black (one) or white (zero). In the setting $\boldsymbol{I_x} = (x_1, x_2, ..., x_d) \in \mathbb{R}^{d=MN}$, M and N are respectively the width and the height of the image sample $\boldsymbol{I_x}$.
In this framework, Wang et al. restrict the design scope to the function g_{ij}, where $\boldsymbol{G} = (g_{ij})_{dxd}$ using equation (A.6). According to the authors[59, 60], a sufficient condition to PSD is that:

(i) The metric $g_{ij} = f(|\boldsymbol{P}_i - \boldsymbol{P}_j|)$ i.e. is a function of the distance between two cells.

(ii) f is continuous, and g_{ij} decreases monotonically as $|\boldsymbol{P}_i - \boldsymbol{P}_j|$ increases.

(iii) f must be a continuous positive definite function.

Wang et al. finally introduce their definition for g_ij using "the most important" positive definite function that respects the sufficient conditions (i),(ii) and (iii) the authors defined to build a PSD matrix. The Gaussian fucntion $G(0, \sigma^2)$[2]:

$$g_{ij} = f(|\boldsymbol{P}_i - \boldsymbol{P}_j|) = \frac{1}{2\pi\sigma^2} e^{-\frac{|\boldsymbol{P}_i - \boldsymbol{P}_j|^2}{2\sigma^2}} \tag{A.7}$$

Fixing $\sigma = 1$ the final Mahalanobis distance metric for Wang et al. becomes:

$$D^2_{G(0,\sigma^2)}(\boldsymbol{I_x}, \boldsymbol{I_y}) = \frac{1}{2\pi} \sum_{i,j=1}^{MN} e^{-\frac{|\boldsymbol{P}_i - \boldsymbol{P}_j|^2}{2}} (x_i - y_i)(x_j - y_j) \tag{A.8}$$

[1] $D(x, z) \leq D(x, y) + D(y, z)$

[2] this is not precisely a normal distribution, because of the constant term in front of the exponent. It is a rescaled version of a normal distribution with mean $\mu = 0$ and $sigma^2$ that we call G.

This choices produces the improvements presented in the following picture:

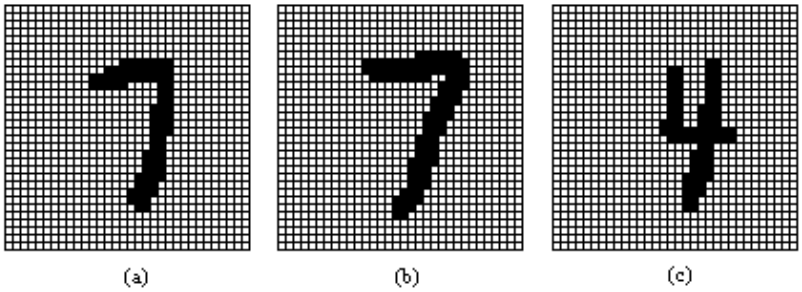


Figure A.1: A screenshot showing three different binary images of 32x32 pixels each. The result of the usual euclidean distance are: $D_E(a, b) = 54$ and $D_E(a, c) = 49$. The result of Mahalanobis distance discussed are instead: $D_{G(0,\sigma^2)}(a, b) = 23.3$ and $D_{G(0,\sigma^2)}(a, c) = 27.3$ showing the improvement of this approach.

Bibliography

[1] Mark Hendrikx, Sebastiaan Meijer, Joeri Velden, and Alexandru Iosup. «Procedural Content Generation for Games: A Survey». In: *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)* 9 (Feb. 2013). DOI: 10.1145/2422956.2422957 (cit. on pp. 3, 9).

[2] Carolina Islas Sedano, Maira Carvalho, Nicola Secco, and C. Longstreet. «Collaborative and cooperative games: Facts and assumptions». In: May 2013, pp. 370–376. ISBN: 9781467364041. DOI: 10.1109/CTS.2013.6567257 (cit. on p. 4).

[3] Jose Zagal and Jochen Rick. «Collaborative games: Lessons learned from board games». In: *Simulation & Gaming - Simulat Gaming* 37 (Mar. 2006), pp. 24–40. DOI: 10.1177/1046878105282279 (cit. on p. 4).

[4] Neville Stanton, Alan Hedge, Karel Brookhuis, Eduardo Salas, and Hal Hendrick. *Handbook of Human Factors and Ergonomics Methods*. 2004. Chap. 48, pp. 398–399. DOI: 10.1201/9780203489925 (cit. on p. 5).

[5] Scott I. Tannenbaum, Rebecca L. Beard, and Eduardo Salas. «Chapter 5 Team Building and its Influence on Team Effectiveness: an Examination of Conceptual and Empirical Developments». In: *Issues, Theory, and Research in Industrial/Organizational Psychology*. Ed. by Kathryn Kelley. Vol. 82. Advances in Psychology. North-Holland, 1992, pp. 117–153. DOI: https://doi.org/10.1016/S0166-4115(08)62601-1. URL: http://www.sciencedirect.com/science/article/pii/S0166411508626011 (cit. on p. 5).

[6] Noor Shaker, Julian Togelius, and Mark J. Nelson. *Procedural Content Generation in Games*. 1st. Springer Publishing Company, Incorporated, 2016. Chap. 1. ISBN: 3319427148 (cit. on pp. 5, 9, 10).

[7] Smith Gillian. «An Analog History of Procedural Content Generation». In: *FDG*. 2015 (cit. on pp. 5, 9).

[8] Francois Pachet. «Beyond the Cybernetic Jam Fantasy: The Continuator». In: *IEEE computer graphics and applications* 24 (Jan. 2004), pp. 31–5. DOI: 10.1109/MCG.2004.1255806 (cit. on p. 5).

[9] Nuno Barreto, Amílcar Cardoso, and Licinio Roque. «Computational Creativity in Procedural Content Generation: A State of the Art Survey». In: Nov. 2014. DOI: 10.13140/2.1.1477.0882 (cit. on p. 5).

[10] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne. «Search-Based Procedural Content Generation: A Taxonomy and Survey». In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3 (2011), pp. 172–186 (cit. on pp. 6, 9, 10, 12).

[11] Barton Matt and Loguidice Bill. *The History of Rogue: Have @ You, You Deadly Zs*. 2009. URL: https://www.gamasutra.com/view/feature/132404/the_history_of_rogue_have__you_.php (cit. on p. 6).

[12] Spufford Francis. *Masters of their universe, edited extract from Backroom Boys: The Secret Return Of The British Boffin*. 2013. URL: https://www.theguardian.com/books/2003/oct/18/features.weekend (cit. on p. 7).

[13] Trent Ward. *Diablo for PC review*. 1997. URL: https://www.gamespot.com/reviews/diablo-review/1900-2538662/ (cit. on p. 7).

[14] Cindy Yans. *Diablo*. 1997. URL: https://web.archive.org/web/20030710195138/http://www.cdmag.com/articles/002/055/diablo_review.html (cit. on p. 7).

[15] Richard Moss. *7 uses of procedural generation that all developers should study*. 2016. URL: https://www.gamasutra.com/view/news/262869/7_uses_of_procedural_generation_that_all_developers_should_study.php (cit. on p. 7).

[16] Gillian Smith. «Understanding procedural content generation: A design-centric analysis of the role of PCG in games». In: *Conference on Human Factors in Computing Systems - Proceedings* (Apr. 2014). DOI: 10.1145/2556288.2557341 (cit. on p. 7).

[17] *Civilization Fanatics Center*. https://www.civfanatics.com/civ4/ (cit. on p. 7).

[18] Chris Hecker, Bernd Raabe, Ryan W. Enslow, John DeWeese, Jordan Maynard, and Kees van Prooijen. «Real-time Motion Retargeting to Highly Varied User-Created Morphologies». In: *Proceedings of ACM SIGGRAPH '08*. http://chrishecker.com/Real-time_Motion_Retargeting_to_Highly_Varied_User-Created_Morphologies. 2008 (cit. on p. 7).

[19] *Borderlands Weapons*. https://borderlands.fandom.com/wiki/Weapons (cit. on p. 7).

[20] Wikipedia contributors. *List of best-selling video games — Wikipedia, The Free Encyclopedia*. 2020. URL: https://en.wikipedia.org/w/index.php?title=List_of_best-selling_video_games%5C&oldid=950274274 (cit. on p. 7).

[21] Markus Persson. *Terrain generation, Part 1*. 2011. URL: https://notch.tumblr.com/post/3746989361/terrain-generation-part-1 (cit. on p. 7).

[22] Richard Cobbett. *From shareware superstars to the Steam gold rush: How indie conquered the PC*. 2017. URL: https://www.pcgamer.com/from-shareware-superstars-to-the-steam-gold-rush-how-indie-conquered-the-pc/ (cit. on p. 8).

[23] Wikipedia contributors. *Metagaming — Wikipedia, The Free Encyclopedia*. 2020. URL: https://en.wikipedia.org/w/index.php?title=Metagaming%5C&oldid=946990764 (cit. on p. 8).

[24] Mark Johnson. *Before Spelunky and FTL, There Was Only ASCII*. 2015. URL: https://www.pastemagazine.com/games/before-spelunky-and-ftl-there-was-only-ascii/ (cit. on p. 8).

[25] *Gamasutra's Best of 2016: The top 10 game developers of the year*. 2016. URL: https://www.gamasutra.com/view/news/287153/Gamasutras_Best_of_2016_The_top_10_game_developers_of_the_year.php (cit. on p. 8).

[26] Matt Peckham. *No Man's Sky is wildly ambitious, utterly vast and a huge challenge to the video game industry's status quo*. 2016. URL: https://time.com/no-mans-sky/ (cit. on p. 8).

[27] Alexandru Iosup. «POGGI: Puzzle-Based Online Games on Grid Infrastructures». In: vol. 5704. Jan. 2009, pp. 390–403. DOI: 10.1007/978-3-642-03869-3_39 (cit. on p. 9).

[28] J. Togelius, A.J. Champandard, Pier Luca Lanzi, M. Mateas, A. Paiva, Mike Preuss, and K.O. Stanley. «Procedural content generation: goals, challenges and actionable steps». In: *Dagstuhl Follow-Ups* 6 (Jan. 2013), pp. 61–75 (cit. on p. 10).

[29] Benjamin van Arkel, Daniel Karavolos, Anders J. Bouwer, Sander Bakkes, and Frank Nack. «Procedural generation of collaborative puzzle-platform game levels». In: 2015 (cit. on p. 13).

[30] Aliona Kozlova, Joseph Brown, and Elizabeth Reading. «Examination of Representational Expression in Maze Generation Algorithms». In: Aug. 2015. DOI: 10.1109/CIG.2015.7317902 (cit. on pp. 16, 17).

[31] Peter Gabrovšek. «Procedural Content Generation for Games: A Survey». In: *The IPSI BgD Transactions on Internet Research* 15 (Jan. 2019), pp. 26–33 (cit. on pp. 16, 17).

[32] Albin Karlsson. *Evaluation of the Complexity of Procedurally Generated Maze Algorithms*. 2018 (cit. on pp. 16, 18).

[33] Martin Foltin. «Automated Maze Generation and Human Interaction». In: 2011 (cit. on pp. 16, 17).

[34] Pullen D. Walter. *Maze classification*. June 2015. URL: http://www.astrolog.org/labyrnth/algrithm.htm (cit. on pp. 18, 19, 25).

[35] Eric W. Weisstein. «Cellular Automaton». In: *MathWorld–A Wolfram Web Resource* (). URL: https://mathworld.wolfram.com/CellularAutomaton.html (cit. on p. 19).

[36] *Cellular Automaton*. 2009. URL: https://www.conwaylife.com/wiki/Cellular_automaton (cit. on pp. 19, 20).

[37] Peak David, West Jevin D., Messinger Susanna M., and Mott Keith A. «Evidence for complex, collective dynamics and emergent, distributed computation in plants». In: *Proceedings of the National Academy of Sciences of the United States of America* 101 (Jan. 2004), pp. 918–922. DOI: 10.1073/pnas.0307811100 (cit. on p. 19).

[38] M. Gerhardt and H. Schuster. «A cellular automaton describing the formation of spatially ordered structures in chemical systems». In: *Physica D: Nonlinear Phenomena* 36.3 (1989), pp. 209–221. ISSN: 0167-2789. DOI: https://doi.org/10.1016/0167-2789(89)90081-X. URL: http://www.sciencedirect.com/science/article/pii/016727898990081X (cit. on p. 19).

[39] D. R. Chowdhury, S. Basu, I. S. Gupta, and P. P. Chaudhuri. «Design of CAECC - cellular automata based error correcting code». In: *IEEE Transactions on Computers* 43.6 (1994), pp. 759–764 (cit. on p. 19).

[40] Forsyth T. *Cellular Automata for Physical Modelling*. Charles Eiver Media, Inc., 2002 (cit. on p. 19).

[41] Lawrence Johnson, Georgios Yannakakis, and Julian Togelius. «Cellular automata for real-time generation of». In: (Sept. 2010). DOI: 10.1145/1814256.1814266 (cit. on pp. 19, 22).

[42] Roland Linden, Ricardo Lopes, and Rafael Bidarra. «Procedural Generation of Dungeons». In: *Computational Intelligence and AI in Games, IEEE Transactions on* 6 (Mar. 2014), pp. 78–89. DOI: 10.1109/TCIAIG.2013.2290371 (cit. on p. 22).

[43] Alain Fournier, Donald Fussell, and Loren Carpenter. «Computer Rendering of Stochastic Models». In: *Commun. ACM* 25 (June 1982), pp. 371–384. DOI: 10.1145/358523.358553 (cit. on p. 23).

[44] Gavin S. P. Miller. «The definition and rendering of terrain maps». In: *SIGGRAPH '86*. 1986 (cit. on p. 23).

[45] Ken Perlin. «An Image Synthesizer». In: *Computer Graphics, Vol. 19, No. 3.* 1985, pp. 287–296 (cit. on p. 23).

[46] Ken Perlin. «Noise hardware. In Real-Time Shading SIGGRAPH Course Notes». In: *Real-Time Shading SIGGRAPH Course Notes.* 2001. URL: http://www.csee.umbc.edu/~olano/s2002c36/ch02.pdf (cit. on p. 23).

[47] David S. Olton and Robert J. Samuelson. «Remembrance of Places Passed: Spatial Memory in Rats.» In: 1976 (cit. on p. 24).

[48] D. C. Dracopoulos. «Robot path planning for maze navigation». In: *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227).* Vol. 3. 1998, 2081–2085 vol.3 (cit. on p. 24).

[49] Gene Jan, Ki Yin Chang, and Ian Parberry. «A new maze routing approach for path planning of a mobile robot». In: Aug. 2003, 552–557 vol.1. ISBN: 0-7803-7759-1. DOI: 10.1109/AIM.2003.1225154 (cit. on p. 24).

[50] Vladimir J. Lumelsky and Senior Member. «A comparative study on the path length performance of maze searching and robot motion planning algorithms». In: *IEEE Trans. on Robotics and Automation* (1991) (cit. on p. 24).

[51] Nageswara Rao and Sundararaj Iyengar. «Autonomous Robot Navigation in Unknown Terrains: Incidental Learning and Environmental Exploration». In: *Systems, Man and Cybernetics, IEEE Transactions on* 20 (Dec. 1990), pp. 1443–1449. DOI: 10.1109/21.61213 (cit. on p. 24).

[52] Narayan K. Lalit, Rao K. Mallikarjuna, and Sarcar M.M.M. *Computer Aided Design and Manufacturing.* 1st. New Delhi: Prentice Hall of India, 2008. Chap. 1. ISBN: 9788120333420 (cit. on pp. 28, 29).

[53] Todd Lubart. «How Can Computers be Partners in the Creative Process: Classification and Commentary on the Special Issue». In: *International Journal of Human-Computer Studies* 63 (Oct. 2005), pp. 365–369. DOI: 10.1016/j.ijhcs.2005.04.002 (cit. on p. 28).

[54] Antonios Liapis, Gillian Smith, and Noor Shaker. «Chapter 11 Mixed-initiative content creation». In: 2016 (cit. on p. 28).

[55] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. «Sentient Sketchbook: Computer-aided game level authoring». In: *FDG.* 2013 (cit. on p. 29).

[56] S.J. Russell, S.J. Russell, P. Norvig, and E. Davis. *Artificial Intelligence: A Modern Approach.* 3rd. Prentice Hall, 2010. ISBN: 9780136042594. URL: https://books.google.it/books?id=8jZBksh-bUMC (cit. on pp. 30, 44).

[57] Lague Sebastian. *Create a Game series*. 2015. URL: https://forum.unity.com/threads/create-a-game-from-scratch-7hr-tutorial-series.355732/ (cit. on p. 35).

[58] Herman Tulleken. *Algorithms for making more interesting mazes*. 2016. URL: https://www.gamasutra.com/blogs/HermanTulleken/20161005/282629/Algorithms_for_making_more_interesting_mazes.php (cit. on pp. 38, 41, 42).

[59] Liwei Wang, Yan Zhang, and Jufu Feng. «On the Euclidean distance of images». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.8 (2005), pp. 1334–1339 (cit. on pp. 45, 46, 68, 69).

[60] Bing Sun, Jufu Feng, and Liwei Wang. «Learning IMED via shift-invariant transformation». In: June 2009, pp. 1398–1405. DOI: 10.1109/CVPR.2009.5206720 (cit. on pp. 46, 68, 69).

[61] Nathan Sorenson and Philippe Pasquier. «The evolution of fun: Automatic level design through challenge modeling». In: (Jan. 2010) (cit. on pp. 47, 48).

[62] Nathan Sorenson and Philippe Pasquier. «Towards a Generic Framework for Automated Video Game Level Creation». In: vol. 6024. Apr. 2010, pp. 131–140. DOI: 10.1007/978-3-642-12239-2_14 (cit. on pp. 47, 48).

[63] Nathan Sorenson, Philippe Pasquier, and Steve Dipaola. «A Generic Approach to Challenge Modeling for the Procedural Creation of Video Game Levels». In: *IEEE Trans. Comput. Intellig. and AI in Games* 3 (Sept. 2011), pp. 229–244. DOI: 10.1109/TCIAIG.2011.2161310 (cit. on pp. 47, 48).

[64] Gillian Smith, Mee Cha, and Jim Whitehead. «A framework for analysis of 2D platformer levels». In: *Sandbox '08*. 2008 (cit. on p. 47).

[65] Paolo Piselli, Mark Claypool, and James Doyle. «Relating Cognitive Models of Computer Games to User Evaluations of Entertainment». In: Jan. 2009, pp. 153–160. DOI: 10.1145/1536513.1536545 (cit. on p. 48).

[66] Mark J. Nelson. «Game Metrics Without Players: Strategies for Understanding Game Artifacts». In: *Artificial Intelligence in the Game Design Process*. 2011 (cit. on p. 50).

[67] Prasanta Chandra Mahalanobis. «On the generalized distance in statistics». In: *Proceedings of the National Institute of Sciences (Calcutta)* 2 (1936), pp. 49–55 (cit. on p. 68).

[68] Marc Law. «Distance Metric Learning for Image and Webpage Comparison». In: (Jan. 2015), pp. 13–15 (cit. on p. 68).