UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

LUÍS FERNANDO MAIA SANTOS SILVA

BALANCING AND TRANSPOSITION OF MAPS FOR LOCATION-BASED GAMES

FORTALEZA

2019

LUÍS FERNANDO MAIA SANTOS SILVA

BALANCING AND TRANSPOSITION OF MAPS FOR LOCATION-BASED GAMES

Thesis submitted to the Programa de Pós-Graduação em Ciência da Computação of the Universidade Federal do Ceará, in partial fulfillment of the requirements for the degree of Doctor in Computer Science. Concentration Area: Software Engineering

Supervisor: Prof. Dr. Fernando Antonio Mota Trinta

Co-Supervisor: Prof. Dr. Windson Viana de Carvalho

FORTALEZA

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S581b Silva, Luís Fernando Maia Santos.
Balancing and Transposition of Maps for Location-based Games / Luís Fernando Maia Santos Silva. – 2019.
155 f. : il. color.

Tese (doutorado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação , Fortaleza, 2019.
Orientação: Prof. Dr. Fernando Antonio Mota Trinta.
Coorientação: Prof. Dr. Windson Viana de Carvalho.

1. Location-based Games. 2. Procedural Content Generation. 3. Graph Isomorphism. I. Título.
CDD 005

LUÍS FERNANDO MAIA SANTOS SILVA

BALANCING AND TRANSPOSITION OF MAPS FOR LOCATION-BASED GAMES

Thesis submitted to the Programa de Pós-Graduação em Ciência da Computação of the Universidade Federal do Ceará, in partial fulfillment of the requirements for the degree of Doctor in Computer Science. Concentration Area: Software Engineering

Approved in: July 11, 2019

EXAMINATION COMMITTEE

Prof. Dr. Fernando Antonio Mota Trinta (Supervisor)
Federal University of Ceará (UFC)

Prof. Dr. Windson Viana de Carvalho (Co-Supervisor)
Federal University of Ceará (UFC)

Prof. Dr. Esteban Walter Gonzalez Clua
Fluminense Federal University (UFF)

Profa. Dra. Emanuele Marques dos Santos
Federal University of Ceará (UFC)

Prof. Dr. Rudini Menezes Sampaio
Federal University of Ceará (UFC)

Prof. Dr. Cláudio de Souza Baptista
Federal University of Campina Grande (UFCG)

I dedicate this work to Tainá, who spent such a short time among us, but certainly made our lives happier.

ACKNOWLEDGMENTS

First, I would like to thank my advisors Prof. Fernando Antonio Mota Trinta and Prof. Windson Viana de Carvalho for accepting me as one of their students, for guiding me through every step of this work, and for supporting me when I could not dedicate myself to this research in full time. The past 2 years have been a very difficult and labor intensive period that I could not have endured without your help and patience. Thank you very much.

In my opinion, the doctorate is more about a journey than a title. Thus my previous advisors Prof. Francisco Vieira de Souza (UFPI) and Prof. João Luiz Dihl Comba (UFRGS) are also to be thanked for their contribution to shape me as a researcher during graduation and masters, respectively.

I would like to highlight my gratitude to all colleagues of Federal University of Ceará (UFC) in the name of Paulo Artur Duarte, for being one of the most altruistic persons I had the pleasure to work with, and also thank Prof. Rossana Maria de Castro Andrade on behalf of all the professionals that make up the Group of Computer Networks, Software Engineering and Systems (GREat). What a wonderful research team.

Thanks to all my colleagues from the Federal Institute of Education of Maranhão (IFMA), for allowing me to spend two years in full time dedication to the doctorate, and for discussions and suggestions about this work. I also must thank prof. João da Paixão Soares for encouraging me to start this journey when it seemed altogether unfeasible.

A special thanks goes to my wife Soraya, for agreeing to suddenly change her life to follow me along this phase, to my brothers André and Isaias, who are my true friends, and my parents Joseane Maia and Luís Faustino, that have spared no effort to provide me the best education possible. You are the true inspiration to my life and career.

“Codes are a puzzle. A game, just like any other game.”

(Alan Turing)

RESUMO

Jogos-baseados em Localização (JBLs) são aqueles que dependem da localização do jogador como principal traço de jogabilidade para alterar seu estado de jogo. Por isso, desenvolver JBLs que estejam disponíveis em todo o mundo é uma tarefa desafiadora que requer o desenvolvimento de instâncias dos jogos em vários locais, mantendo o mesmo balanceamento, recursos e até mesmo correlações entre os locais do jogo e o mundo real. Logo, é praticamente impossível projetar manualmente interações, desafios e cenários para cada local em que um jogador está. Portanto, geralmente o mesmo jogo apresenta instâncias distintas com níveis de dificuldade variados devido a diferenças de terreno, distância, disponibilidade de transporte etc. Consequentemente, até mesmo empresas estabelecidas no mercado possuem dificuldade para implantar JBLs que estejam disponíveis em todo o mundo. Logo, os atuais JBLs não estão disponíveis em muitas regiões, especialmente cidades pequenas e bairros pobres das grandes cidades. Além disso, estes jogos ainda apresentam enormes diferenças de balanceamento entre localidades e evitam explorar a competição entre jogadores, como em outros gêneros de jogos. Nesta tese, propomos um método de transposição de mapas de JBLs, com foco na manutenção do seu balanceamento. Esta abordagem depende de informações sobre Pontos de Interesse (POIs) em torno da localização dos jogadores e estimativas sobre o custo de movimentação entre estes pontos. Introduzimos duas medidas para estimar o balanceamento em JBLs modernos e implementamos três algoritmos diferentes que visam a transposição de seus mapas com variações mínimas no seu balanceamento. A primeira medida avalia o balanceamento de jogos internamente e a segunda compara o balanceamento entre duas instâncias de um jogo. Neste caso, propomos converter os jogos em grafos ponderados direcionados e utilizar um dos algoritmos para gerar uma instância equivalente, de acordo com a localização do jogador. Para validar a abordagem proposta, projetamos quatro JBLs distintos em termos de recurso, jogabilidade e mecânica, e conduzimos um experimento com usuários para comparar mapas gerados por esses algoritmos em diferentes locais. Os resultados indicam que os jogos com balanceamento semelhante apresentaram pontuação mais alta, e que os algoritmos apresentam diferente desempenho conforme o número de POIs. Finalmente, podemos concluir que este trabalho contribui para melhorar o desenvolvimento de JBLs, ajudando a mitigar o desafio da transposição balanceada.

Palavras-chave: Jogos Baseados em Localização. Geração Procedural de Conteúdo. Isomorfismo de grafos.

ABSTRACT

Location-Based Games (LBGs) rely on the player's location to change its game state, usually as the main trait of playability. Thus, developing worldwide LBGs is a challenging task due to the need to deploy game instances in multiple locations, while maintaining the same game balancing, features, and even correlations between locations of the game and the real world. Since LBGs rely on players' location, it is virtually impossible to manually design interactions, challenges, and game scenarios for every place a player is at. Therefore, the same LBG is likely to have distinct instances with varying difficulty levels because of differences in terrain, distance, transport availability, etc. As a result, even established game companies struggle to deploy LBGs around the globe, so the current generation of LBGs is not available in many areas, especially small cities and poor neighborhoods of big cities. Additionally, modern LBGs still present huge balancing differences between regions and avoid exploring the competition between players like other game genres. In this thesis, we propose a method for transposing LBGs maps while focusing on maintaining their game balancing. This approach depends on information about Points-of-Interest (POIs) around the players' location and estimations about the cost to move between POIs. We introduced two measurements to estimate game balancing in modern LBGs and implemented three different algorithms that aim at transposing LBGs' maps with minimal variations in game balancing. The first measurement, called *Internal Balancing Difference*, assesses game balancing internally and the second, called *Minimum Balancing Difference*, compares game balancing between two instances of a game. The transposition algorithms are based on the Monte Carlo tree search, the Ullmann's algorithm, and Genetic Algorithms. In this case, we convert LBGs into directed weighted graphs and use one of the algorithms to generate an LBG instance according to the player's location. To validate the proposed approach, we designed four LBGs with distinct features, gameplay, and mechanics, and conducted an experiment that required samples to compare maps generated by these algorithms in different locations. Results indicate that games with similar game balancing score higher and that the algorithms differ in performance depending on the number of POIs. Finally, we can conclude that this work contributes to improve the development of LBGs by helping to mitigate the challenge of transposing LBGs while maintaining game balancing.

Keywords: Location-based Games. Procedural Content Generation. Graph Isomorphism.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF SOURCE-CODE

LIST OF ABBREVIATIONS

API	Abstract Programming Interface
ASP	Answer Set Programming
BPMN	Business Process Model Notation
EUD	End User Development
GA	Genetic Algorithm
GMP	Graph Matching Problem
GPS	Global Positioning System
GPU	Graphics Processing Unit
GREat	Group of Computer Networks, Software Engineering and Systems
IoT	Internet of Things
LBG	Location-based Game
MCTS	Monte Carlo Tree Search
PBF	Parallel Brute-Force
PCG	Procedural Content Generation
PDA	Personal Digital Assistant
POIs	Points of Interest
PWU	Parallel Weighted Ullmann
RTS	Real-Time Strategy
TSP	Traveling Salesman Problem
UCT	Upper Confidence Bounds for Trees
WGMP	Weighted Graph Matching Problem
WSM	Weighted Subgraph Matching

LIST OF SYMBOLS

$\mathfrak{I}$	Internal Difficulty Level
$\mathfrak{M}$	Minimum Balance Difference
$\mathfrak{T}$	Cost function
$\mathfrak{S}$	Grand sum of a matrix
κ	Percentage of solutions considered good
$\mathscr{B}$	Budget for the MCTS algorithm
τ	Threshold used in the PWU algorithm
δ	Factor by which the τ is reduced
$\mathscr{M}$	Maximum number of solutions for the PWU to process
ψ	Size of population used in the GA algorithm
η	Number of generations to evolve
ε	Elitism factor
ρ	Quantity of chromosomes in tournament
μ	Probability of a mutation happening

SUMMARY

1 INTRODUCTION

This thesis presents a study and methods to tackle the challenge of transposing maps of LBGs while focusing on maintaining game balancing. Hence, this work proposes measurements to gauge the difficulty level and balancing of LBGs, and a method that takes a game and a location as input to create a transposed instance of the game's map that minimizes the differences in game balancing. Furthermore, three algorithms that tackle this challenge as a Weighted Graph Matching Problem (WGMP) were developed.

This chapter introduces this thesis and is organized as follows. Section 1.1 describes the context in which this research is applied, while Section 1.2 presents the motivation for the development of this work and the challenges addressed by the proposed method. Section 1.3 presents the hypothesis that was tested and research questions that guided the experiments. Next, Section 1.4 details the goals and contributions of this work, and Section 1.5 presents the methodology used in this research. Lastly, Section 1.6 describes the structure of this thesis.

1.1 Context

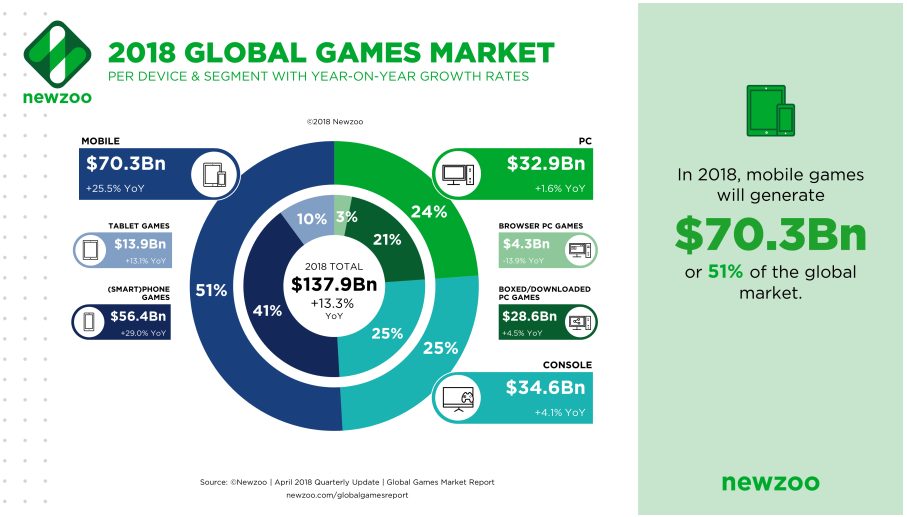
Mobile devices are best known for their mobility and for providing quick access to the Internet. The release of the first iPhone in 2007 marked a new era in the mobile device market. Since then, the smartphone has become a technological hub that includes several sensors, considerable processing power, and is capable of countless applications.

In the last decade, the expansion of this new market by traditional manufacturers has lead to the dissemination of smartphones in practically every country of the globe. As a result, the popularization of smartphones and tablets has caused, along with the Internet, a technological revolution responsible for changing the behavior, culture and social habits around the world.

In a short time, many companies realized that smartphones and tablets could become a platform for playing games. Previously, games were played in specific places using devices such as keyboard, mouse, joystick, PCs, and consoles (NICKLAS *et al.*, 2001). Nowadays, smartphones have enabled games to be played at any time and everywhere. Consequently, many games have been ported or developed specifically to the mobile market. To illustrate the importance of mobile devices to the game industry, Newzoo -a global game and mobile market research company- released a report in April 2018 stating that *"mobile games will continue to be the largest segment following 10 years of double-digit growth"* (NEWZOO, 2018). It also

claims that, for the first time, more than half of all game revenues will come from the mobile segment, as shown in Figure 1.

Figure 1 – Chart depicting the global game market divided by platform.



Source: (NEWZOO, 2018).

In fact, smartphones have impacted the way people play games and have boosted the game industry due to its immense potential as a gaming platform. Additionally, the increase in processing power and the popularization of sensors in modern smartphones (e.g., accelerometer, compass, camera, Global Positioning System (GPS), gyroscope, among others) has allowed the implementation of new game genres, such as Pervasive Games. Even though the concept of Pervasive Games is not unanimous among researchers, this work uses the definition of McGonigal (2003), that describe these games as "mixed reality" games that use mobile, ubiquitous and embedded digital technologies to create virtual playing fields in everyday spaces (MCGONIGAL, 2003). In summary, these games can have their virtual environment affected by actions and context from the real world.

Given that Pervasive Games include distinct types of games with a plethora of attributes, many researches were conducted in this field exploring game patterns (AKSELSEN; KRISTIANSEN, 2010), features (VALENTE *et al.*, 2018), quality requirements (VALENTE *et al.*, 2017), technologies (NEVELSTEEN, 2015), modelling techniques (GUO, 2015), development methodologies (VIANA *et al.*, 2014), among others.

However, this thesis focuses on a popular subtype of Pervasive Games called Location-based Games (LBGs), which are defined in this work as games that use players' location to modify the game state during runtime. As a result, players have to physically move to progress in the game, thus establishing a link between virtual and real worlds. Depending on the

game space, interface and duration (SILVA; SUTKO, 2009), LBGs are also known as *Urban Games* or *Street Games*.

Lately, both industry and academia have focused efforts on the development of LBGs. Games like *Parallel Kingdom*[1] (released in 2008), *Zombies, Run!*[2] (released in 2012) and *Ingress* (released in 2012), have reached millions of users[3]. Moreover, the release of *Pokémon GO* in 2016 has confirmed this potential as the game has more than 140 million active users, and has generated more than $2 billion dollars in revenue since its launch (SUPERDATA, 2018).

The success of those early LBGs has fostered the development of new titles, such as *Ghostbusters World*, *Jurassic World Alive*, and *The Walking Dead: Our World*, while *Ingress* has received a complete redesign and *Harry Potter: Wizards Unite* was launched in 2019.

1.2 Motivation

Despite the release of more LBGs in the last two years, they still represent just a fraction of the mobile gaming industry. In fact, only a few game studios are currently venturing into this area. Niantic, one of the pioneers in the development of LBGs, has developed more games than any other gaming company. However, it is responsible for only three titles, namely *Ingress*, *Pokémon GO*, and *Harry Potter: Wizards Unite*. Other studios such as FourThirtyThree (*Ghostbusters World*), Ludia (*Jurassic World Alive*), and Next Games (*The Walking Dead: Our World*) have designed one title each, so far. Considering the great success and profit generated by some of these games (e.g. Niantic is reportedly worth more than Square Enix, Capcom, and Sega in the end of 2018, according to The Wall Street Journal[4]), it was expected that more game developers would commit efforts to making new LBGs.

Nevertheless, cost and time are key problems to produce digital games, with many AAA[5] titles taking more than 3 years of development, with costs that exceed tens of millions of dollars. For instance, sources pointed out estimations around $44 million dollars to produce *God of War 3* almost a decade ago (SCHILLE, 2010) and astounding $250 million dollars to the development and marketing of *GTA V* in 2013 (MCLAUGHL, 2013). This scenario is

[1] http://www.parallelkingdom.com/community/update-hut.aspx#177
[2] https://medium.com/@adrianhon/five-years-of-zombies-run-6e090ef3fe4
[3] http://www.technewsworld.com/story/81106.html
[4] https://www.wsj.com/articles/niantic-maker-of-hit-pokemon-go-app-refuels-with-3-9-billion-valuation-11544748877
[5] An informal classification term used by the gaming industry to refer titles with the highest development budgets and levels of promotion. Similarly, to blockbuster movies, a AAA game is therefore expected to be a high quality title or to be among the best sellers.

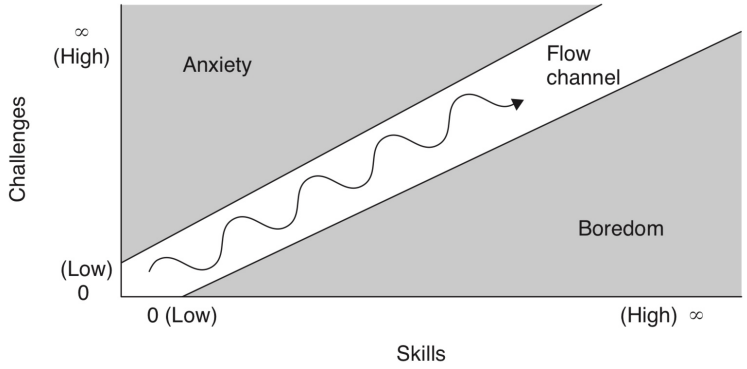
not different for LBGs, with Niantic and Google spending roughly $30 million dollars in the development of *Pokémon GO* (LITTLE, 2016). As a result, game studios are continuously trying to reduce expenses to both minimize risks and increase profits. In some cases, game studios have to lay-off to balance their budgets, as it has been recently reported by Next Games, the company behind the development of games for the *The Walking Dead* franchise (BATCHELOR, 2019).

In addition, producing an LBG is even more challenging because it requires the inclusion of features and dealing with issues that are not present in most mobile games. For instance, developers have to create games to be available in a vast number of places, cope with tracking issues, and adjust game balancing in different regions. Usually, LBGs convert places from the real world to virtual elements of the game, and players have to move to specific locations in order to achieve goals, collect items, encounter opponents, etc. Therefore, an LBG can be played virtually everywhere, if their game elements are correctly mapped to the proper locations in the real world. To address this challenge, a strategy employed by some games is to use a database containing Points of Interest (POIs) scattered around the globe. The main drawbacks of this approach are the complexity, time, and cost to build and maintain such a large database of POIs. However, a low-cost and more feasible solution could involve Procedural Content Generation (PCG), a field of Computer Science that proposes methods and algorithms used in the creation of content automatically, thus reducing costs and time of production.

In addition, maps of LBGs must fulfill requirements that suite aspects of both real and virtual environments, such as avoiding private properties and unreachable areas, and adapting to varying interactions implemented by the games. Furthermore, developers must consider the time and effort to physically move between sites as a crucial factor to game balancing. The definition of game balancing relates to the difficulty level faced by players and is considered by Malone (1981) as one of the three quality factors responsible for the engagement in games, and thus it is crucial to the success of any game (FALSTEIN, 2004). According to Schell (2008), "...if play is too challenging, the player becomes frustrated. But if the player succeeds too easily, they can become bored.", hence developers must balance games in such a way that they keep the experiences of challenge and success in proper balance. To illustrate this duality, Schell (2008) proposed the chart depicted in Figure 2, which indicates that the game difficulty level should fluctuate, but within certain boundaries.

Consequently, two main challenges that hinder the mass production of new LBGs. First, the need to map the game to countless places in such a way the generated instances

Figure 2 – Graphic showing the relation between game balancing and players' attitude towards a good game balancing.



Source: (SCHELL, 2008).

can preserve their game balancing; and second, the high costs and time required to develop worldwide LBGs (TREGEL *et al.*, 2017). It is believed that these challenges have impaired many game studios and independent developers from creating more games. This scenario is the key motivation for this research, and an alternative to alleviate this problem called *Transposition of maps for LBGs* is explored. It consists in using PCG to create maps of LBGs according to the location of each player. Another crucial feature addressed in this thesis concerns the balancing of transposed maps, as a good map transposition has to simultaneously allow the game to be played in multiple places and provide equivalent balancing in every instance of the game.

1.3 Hypothesis and Research Questions

Considering the previously described scenario, this thesis investigates the following hypothesis: "It is possible to create a PCG method that transposes maps of LBGs while preserving their game balancing?"

From this hypothesis, the following Research Questions (RQ) are proposed:

- **RQ1** How to gauge or estimate game balancing in LBGs? *Rationale: Before attempting to design a method for transposing maps of LBGs, it is necessary to gauge its game balancing. Such measurement is key to assess the effectiveness of the transposition in generating balanced game instances.*
- **RQ2** How to automatically transpose maps of LBGs to multiple places while enforcing the same balancing level among all instances? *Rationale: To conduct a successful transposition it is important not only to generate an instance of the game where the player is, but to deliver similar balancing for each step of the game. Consequently, this question requires*

the development of approaches that address both transposition and balancing challenges simultaneously.

- **RQ3** How to validate the transposition of maps of LBGs in terms of correctness and efficiency in maintaining game balancing? *Rationale: It is necessary to ensure the transposition method is working properly both in terms of creating a valid instance of the game map, but also ensuring the game balancing defined in the original game is preserved.*

1.4 Goals and Contribution

There are many researches developed in the field of LBGs, including game design (MAIA *et al.*, 2016; O'HARA, 2008; NEVELSTEEN, 2015; HOLM; LAURILA, 2014; CARMOSINO *et al.*, 2017; ROUNGAS; DALPIAZ, 2015), authoring tools (SILVA *et al.*, 2017, 2017; MALEGIANNAKI; DARADOUMIS, 2017; WETZEL *et al.*, 2012), map balancing and game transposition (MAIA *et al.*, 2017; MACVEAN *et al.*, 2011; FERREIRA *et al.*, 2019), modeling (FERREIRA *et al.*, 2017), usability and experience evaluation (PAAVILAINEN *et al.*, 2017; SÖBKE *et al.*, 2017), frameworks and development techniques (VALENTE; FEIJÓ, 2014; GUO *et al.*, 2015; LOCHRIE *et al.*, 2013; REID, 2008) or introducing games for specific purposes, such as tourism (GUARDIA *et al.*, 2012; ARKENSON *et al.*, 2014), health (WITKOWSKI, 2013; STANLEY *et al.*, 2010), education (LUND *et al.*, 2011; FLINTHAM *et al.*, 2011), culture (RUBINO *et al.*, 2015; CHANG *et al.*, 2014), etc. Nevertheless, there could not be found works that focus on a broad and general approach for transposing maps of LBGs yet.

In this case, this thesis aims at *devising PCG approaches that can assist in the development of LBGs by automatizing the transposition of their maps to multiple locations. Moreover, the transposition has to prioritize the game balancing, so the transposed instances will deliver a similar difficulty level*. While the transposition is key to ensure game availability, the latter is equally important since game balancing is fundamental to keep players engaged in the game.

As a result, this work is expected to achieve the following contributions:

- Design of a general model that can represent several types of LBGs into a simplified data structure containing the essential information for performing a balanced map transposition.
- Devise measurements that can estimate the balancing level of an LBG map.
- Develop PCG methods for the transposition of maps of LBGs to multiple locations while preserving game balancing.

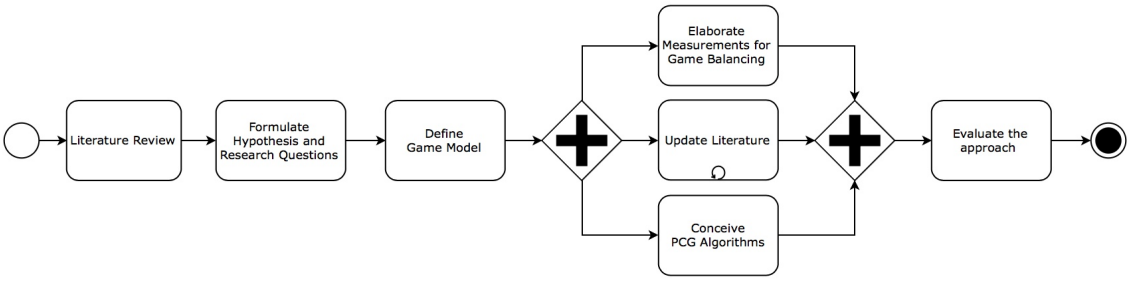
- Validate the proposed approach with several subjects and in multiple locations.

It is important to highlight that the model designed to represent LBGs may not suit all types of mechanics and games, hence, although the map transposition method is generic, it can only be applied to games that can be converted to the proposed game model.

1.5 Methodology

This work consists of an Applied research with Explanatory purpose. Its hypothesis was tested using Quantitative approaches and Experimental methods following a 7-step methodology depicted in Figure 3. Next, the activities and outcomes of each step in the methodology will be detailed.

Figure 3 – Overview of the research methodology used in this work.



Source: Elaborated by the author using Business Process Model Notation (BPMN).

First, a literature review was conducted on topics such as *Location-based Games*, *Transposition of LBGs*, *Game Balancing*, and *PCG*. The review consisted in the analysis of approximately 70 papers and resulted in the study of 38 LBGs. This phase was fundamental to identify the challenges addressed in this work, to formulate the hypothesis and research questions, and to define a game model capable of representing most LBGs.

In the second step the hypothesis was produced and laid the foundation for the development of three research questions. Consequently, the goals and contributions of this work were also conceived in this step.

The third step consisted of defining the game model that is responsible for making the proposed approach general enough to be applied to most LBGs. The game model is a key component of this research since the proposed method relies on it to estimate the game balancing and to perform the map transposition. In summary, an LBG can have its map transposed using this approach if and only if it can be represented by the game model.

The next three steps are executed in parallel since they rely on the concepts and game

model defined previously. Step 4 relates to the first research question (**RQ1**) and was responsible for devising measurements to estimate game balancing in LBGs. These measurements are key to evaluate the quality of the algorithms developed in this work.

Step 5 consists of updating the literature. Notice that this is a task performed in a loop since it was executed repeatedly during the conception of each game balancing measurement and transposition algorithm. In this case, roughly 40 papers were analyzed, including topics such as *Graph Matching Problem*, *MCTS*, *Parallel Processing*, and *Optimization*.

In the sixth step, three algorithms to perform the balanced transposition of maps of LBGs were developed. These algorithms were conceived to receive as input a game model representing a given LBG and the location the user wants the game map to be transposed to. During processing, the algorithms analyze a set of places in the desired location and select a subset of POIs that present balancing similar to the original game. As a result, the algorithms output a version of the game model with the map transposed, as mentioned in the second research question (**RQ2**).

Finally, the last step consisted of evaluating this approach in terms of performance and quality. Two evaluations were conducted, the first devoted to compare the algorithms using the measurements proposed in Step 4, and the second aiming at validating the proposed approach. This second evaluation validated the proposed approach with several subjects by transposing maps of games to locations previously known, so subjects could assess whether the method and its algorithms work properly (**RQ3**).

1.6 Structure of the Thesis

This chapter presented the context and motivation for the development of this thesis. Additionally, it introduced the hypothesis and research questions that guide the research. Finally, a discussion about the goals and expected contributions of this work was provided.

The remaining of this thesis is organized as follows. Chapter 2 (Background) presents the concepts and definitions that are the basis for the development of this research, including an analysis about modern LBGs, Chapter 3 (Related Work) discusses works that are related to the proposed method, Chapter 4 (Balancing and Transposition of Maps for LBGs) presents the proposed method for representing and transposing maps of LBGs and introduces the measurements developed to gauge game balancing in these games.

Chapter 5 (Algorithms) shows that the transposition challenge can be formulated as a

Graph Matching Problem, and details the algorithms proposed in this work. Chapter 6 (Empirical Evaluation) presents an evaluation conducted to assess the algorithms with varying input sizes, and Chapter 7 (User Evaluation) details an evaluation conducted with users to validate the effectiveness of the proposed method to generate games in multiple places. Lastly, Chapter 8 (Conclusions) concludes the thesis, presents limitations, and points to future works.

2 BACKGROUND

This chapter introduces the theoretical foundation upon which this thesis stands, including definitions about Pervasive Games, LBGs, PCG, Graph Matching Problem (GMP), Transposition and Game Balancing. Additionally, it discusses peculiar aspects present in modern LBGs (e.g. game patterns, forms of interaction, mechanics, etc.), and introduces the algorithms used as the foundation to the methods proposed.

The chapter is organized as follows. Section 2.1 presents the concepts and definitions about the games examined in this work. Section 2.2 introduces the main types of mechanics and game patterns present in LBGs. Section 2.3 discusses the challenges and attributes related to the transposition of maps of LBGs, and Section 2.4 examines game balancing in LBGs. A discussion about the balanced transposition of maps of LBGs in distinct game patterns is presented in Section 2.5, Section 2.6 introduces the field of PCG, and Section 2.7 relates this work to the WGMP. Moreover, Section 2.8 introduces the foundation to the algorithms implemented in this research. Lastly, Section 2.9 finishes this chapter.

2.1 Concepts

To better understand the concept of LBGs, it is necessary to revisit its origin, which derives from the studies carried out in the field of Ubiquitous Computing and Pervasive Computing. In the early 1990s, before the development and popularization of smartphones, the concept of Ubiquitous Computing was introduced by Mark Weiser. According to Weiser, the main feature behind ubiquitous computing is the fact that technology becomes intrinsic in everyday activities, allowing the devices and environment around us to interact naturally and transparently, in such a way that technology would "disappear" from people's perceptions (WEISER, 1991). These studies have contributed to the enhancement of mobile devices, the development of wearable computing equipment, and the spread of wireless communication technologies, which are key features in modern digital games.

2.1.1 Pervasive Games

The definition of Ubiquitous Computing and Pervasive Computing diverges in the literature, though most researchers in the area understand the terms as parallel. Likewise, the concepts of Ubiquitous Games and Pervasive Games are also defined differently across

research papers. Some authors consider that Ubiquitous Games are a subset of Pervasive Games and the other way round, whereas other researchers understand these concepts as synonyms (NIEUWDORP, 2007). This thesis does not aim at discussing the understanding of the terms, which will be treated here as interchangeable. Then, as mentioned in the Section 1.1, this work adopts the concept proposed by McGonigal (2003). Therefore, Pervasive Games can be understood as games whose limits are not restricted to a virtual scenario, being the result of a mix between real and virtual environments, so that the boundaries between these environments no longer exist.

According to Kasapakis and Gavalas (2015), current Pervasive Games can be classified into two generations, according to criteria that include communication, evaluation, player equipment, context-awareness support, information model, and orchestration. This last term can be understood as the need to use people in the game environment in order to help players in specific situations and ensure activities are executed properly.

In general, first generation games use technologies such as GPS, Bluetooth, Personal Digital Assistants (PDAs) and notebooks, but also rely on orchestration and context awareness through external sensors. The second generation encompasses games that use smartphones, 3G, triangulation over mobile networks and GPS. In addition, these games have little or no orchestration and capture context using internal sensors of devices, as shown in Table 1.

Table 1 – Common features found in pervasive games generations according to Kasapakis and Gavalas (2015).

Gen	**Time frame**	**Localization**	**Communication**	**Context**	**Orchestration**	**Player Equipment**
1st	2002-2009	GPS/self reporting/ no localization	WiFi/Bluetooth/ Zigbee	Captured by external sensors	Heavy/Light orchestration actions	Custom equipments, wearable computers, PDAs, feature phones
2nd	2009-2014	GPS/Cell-ID	WiFi/3G/Zigbee	Captured by built-in sensors	Light/No orchestration actions	Smartphones
3rd	2014-onwards	GPS/proximity-based localization/crowdsourcing localization platforms	WiFi/WiFi Direct/4G	Captured by built-in sensors/3rd party we-services	No orchestration actions	Wearables (glasses, smart watches, health bands), smartphones

Source: Kasapakis and Gavalas (2015).

Kasapakis and Gavalas (2015) also mention that there is currently a transition of technologies with the emergence of wearable devices, Internet of Things (IoT), and the popularization of faster mobile networks. They claim that, in the near future, the use of these technologies will lead to a third generation of Pervasive Games. As a result, for the development of this study, only second generation games were considered as they are the latest games developed.

2.1.2 Location-based Games

Currently, several types of games are classified as pervasive, such as smart toys, affective games, augmented tabletop games, mobile games, LBGs, alternate reality games and games with augmented or mixed reality. Considering that LBGs are the focus of this work, the scope of the analyzed games was narrowed to include only those that fit in the following definition:

Definition 2.1.1 *Games that, in some way, use the players' location to modify the state of the game throughout their execution, thus creating a connection between the virtual scenario of the game and the real world through the physical space.*

Consequently, LBGs usually require players to move between locations to allow them to collect items, explore places, accomplish missions, or provide encounters between players in both virtual and real worlds.

2.2 Game Patterns in LBGs

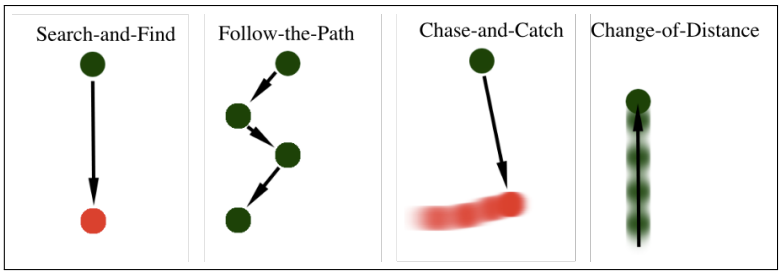
Although there are different types of LBGs, most games share similar features. In general, LBGs monitor either the absolute location of the player or their motion. The former procedure has to establish a connection between locations of the real world and the virtual world, whereas the latter neglects the locations and directions the player chooses to move to, focusing solely on the player's displacement. However, despite the similarity in operation, the form, purpose and sequence with which players move to achieve their goals evidence different patterns of play and types of interaction. In this way, an LBG can make use of one or more game patterns that can even be combined in order to generate more complex interactions. The game patterns present in LBGs were classified by Lehmann (2012) into four distinct types:

- *Search-and-Find*;
- *Follow-the-Path*;
- *Chase-and-Catch*;
- *Change-of-Distance*.

Figure 4 illustrates the main features of these game patterns. It is important to highlight that despite some apparent similarities, these patterns can deliver entirely distinct experiences to players. Conversely, LBGs that implement the same game pattern are, from a

structural point of view, very similar or identical, since they present common implementations, interactions, challenges and traits, even if they are set in contrasting virtual scenarios. Therefore, the analysis of each game pattern is fundamental to solve the problem of transposition in several classes of LBGs. In the following sections, they will be presented in further detail.

Figure 4 – Graphical representation of game patterns for LBGs as classified by Lehmann (2012).



Source: (LEHMANN, 2012)

2.2.1 Search-and-Find

The Search-and-Find pattern is quite popular, and has been implemented in several games. In this pattern, a player needs to find a particular site by moving to it, which can be done with or without the help of a navigation system in the game. In general, LBGs can provide clues that players must follow to find the correct location, or present a set of possible known places so that players can opt for a sequence of explorations.

It is worth mentioning that when using the Search-and-Find game pattern the player is unaware of the exact geographic location of the site being sought, hence players should explore the real world to find the precise place. Another point to be highlighted is that this place will always be a fixed point in the context of the game, so its geographical location is not changed, allowing the player to establish a mapping between real and virtual worlds.

In general, this game pattern is associated with collecting items, and was popularized by the game *Geocaching* (O'HARA, 2008). In this game, the players' main goal is to find objects hidden in certain geographic coordinates, as shown in Figure 5. For the most part, these objects are low value items stored in a box, and once the box is found the player is responsible by replacing one of its items with a new one, so the next player can continue this interaction.

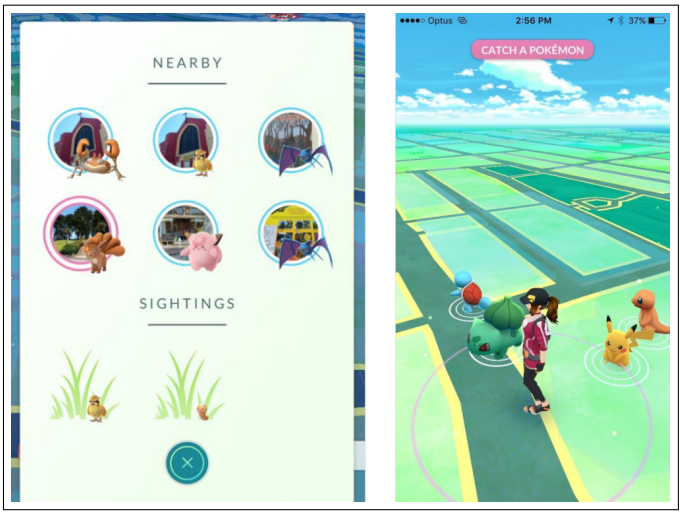
Recently, the *Search-and-Find* pattern has been used as one of the key activities in the game *Pokémon GO*. In this case, players receive tips through the game application informing which pokémons are nearby. Therefore, each player has to explore the real world in search for

Figure 5 – The game indicates the approximate location of objects (left), and an example of a hidden object in the real world (right).

Source: https://www.geocaching.com

the correct geographic locations that hide pokémons (Figure 6).

Figure 6 – App showing pokémons nearby (left) and a map displaying their appearance (right).



Source: Niantic Labs

2.2.2 Follow-the-Path

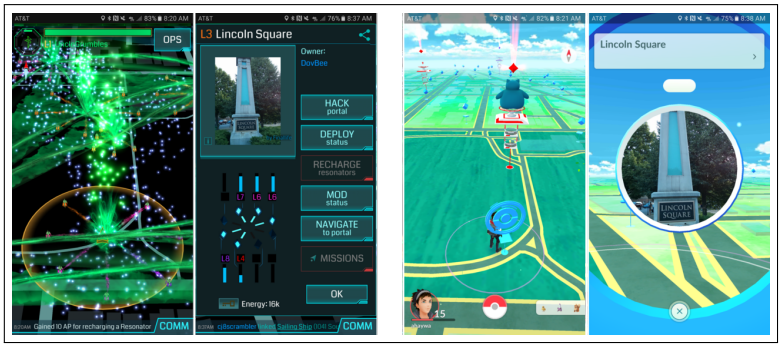
This game pattern demands players to move to certain locations. However, unlike the *Search-and-Find* pattern whose challenge is the discovery of sites, the *Follow-the-Path* pattern dares the player to visit a set of known places. Depending on the goals of the game, it is possible for players to follow multiple paths or incur penalties if they do not follow the predefined routes. For example, in a racing game, players taking shortcuts can be penalized. In tourist applications, it is common for players to follow a specific trail in order to provide information related to the places visited.

The *Follow-the-Path* pattern can also be used to acquire data from the environment, such as to discover the best route between two points. In this case, the locations are presented, but there are no determined paths, therefore players must plan the most efficient path connecting places. Ultimately, the application monitors the players to infer the best route.

Players can also use the *Follow-the-Path* pattern to define target locations in the virtual world as they move around the real world, thus creating new paths for other players to follow. This approach is used in the game *Tourality*, in which a player must complete fixed circuits within a given time, and can also compete with other players (WORKLINE, 2016).

Games such as *Ingress* and *Pokémon GO* also make use of this game pattern, however both games do not impose specific routes to be followed by the players (Figure 7). In both cases, the game plot consists of groups battling each other for the control of certain portals or gymnasiums (POIs), hence it requires players to move to these points in order to conquer them for their group or defend them from the enemy. Consequently, each player chooses the sequence of places to visit, as well as the route to follow.

Figure 7 – Locations are presented to players in *Ingress*(left) and *Pokémon GO*(right).



Source: Niantic Labs

2.2.3 Chase-and-Catch

In the *Chase-and-Catch* pattern, players have to chase a moving element in the game, consequently they must move in the real world to achieve this goal. The chased element may be another player or a virtual character that exists only in the game. In the first case, the LBG works as an adaptation of the traditional game called “Tag”.

The main characteristic of the *Chase-and-Catch* pattern is that the destination players must follow changes recurrently. As a result, the rate at which the destination changes and the capture strategies imposed by the game directly affect the level of difficulty, and thus changes

the game balancing.

This pattern was used in the game *FoxHunt* (MISUND *et al.*, 2009) in both single-player and multiplayer modes. The game places players as fox hunters, so they have to move in the real world chasing foxes displayed on the mobile screen. Eventually, a fox is captured when a hunter manages to approach it in the virtual world. In the game *Shadow Cities*, this game pattern was implemented in a more complex multiplayer environment, as two groups of players battle to capture participants of the opposing group, with each player being simultaneously able to capture and be captured (AREA, 2016).

2.2.4 Change-of-Distance

Unlike the game patterns presented earlier, the *Change-of-Distance* pattern is characterized by disregarding particular destinations. While the other game patterns require players to move in specific directions, in the *Change-of-Distance* pattern the player's destination is irrelevant. In this case, the important interaction is the player's displacement, regardless of destination.

A very popular LBG that uses this pattern is called *Zombies, Run!*, which was developed to stimulate players to perform physical activities. In this game, players are positioned in a virtual scenario full of zombies, which are avoided whenever the player moves in the real world. The game also has a step counter to monitor the player's physical exercises.

The *Change-of-Distance* pattern was also implemented in the game *Pokémon GO* via an activity called egg hatching. In this case, players must walk a certain distance regardless of origin, destination and route, to accomplish the activity.

2.2.5 Mapping Game Patterns to LBGs

In order to investigate the game patterns that are more popular in modern LBGs, an analysis of games regarded as state of the art in the area according to the criteria used by Kasapakis and Gavalas (2015) (Section 2.1.1) was conducted. In general, LBGs developed since 2009 are considered second generation, as they make use of smartphones, 3G, and GPS. As a result, a total of 38 games met the above mentioned criteria from both industry and academia.

Table 2 presents a mapping created by identifying the game patterns present in each LBG. The games were analyzed individually according to their features and interactions. This analysis allowed the recognition of more than one game pattern in some cases.

Table 2 – Table presenting the mapping between the game patterns defined by Lehmann (2012) and 38 second generation LBGs as classified by Kasapakis and Gavalas (2015).

LBGs	***Search-and-Find***	***Follow-the-Path***	***Chase-and-Catch***	***Change-of-Distance***
Geocaching (O'HARA, 2008)	X			
Parallel Kingdom (PATRO *et al.*, 2012)		X		
FoxHunt (MISUND *et al.*, 2009)			X	
Viking Ghost Hunt (CARRIGY *et al.*, 2010)	X			
Hot Potato (CHATZIGIANNAKIS *et al.*, 2010)			X	
Big Game Huntr (LUND *et al.*, 2010)	X			
PiNiZoRo (STANLEY *et al.*, 2010)		X		
Tourality (WORKLINE, 2016)		X	X	
Shadow Cities (AREA, 2016)			X	
The Journey (JAKL, 2004)		X	X	
Exploding Places (FLINTHAM *et al.*, 2011)		X		
Free All Monsters (LUND *et al.*, 2011)		X		
WeQuest (MACVEAN *et al.*, 2011)		X		
O'Munaciedd (GUARDIA *et al.*, 2012)		X		
Treasure (GUO *et al.*, 2012)	X			
Blowtooth (KIRMAN *et al.*, 2012)			X	
See it (NEUSTAEDTER; JUDGE, 2012)	X			
Ingress (LABS, 2019)		X		
Zombies, Run! (WITKOWSKI, 2013)				X
The Walk (START; ALDERMAN, 2016)				X
BattleSuit Runner (SERAPH, 2016)				X
SpecTrek (GAMES4ALL, 2016)		X		
Tidy City Scout (WETZEL *et al.*, 2012)		X		
Floracaching (BOWSER *et al.*, 2013)	X			
Easter Egg Hunt (JORDAN *et al.*, 2013)	X			
Barbarossa (KASAPAKIS *et al.*, 2013)	X		X	
TARX (LOCHRIE *et al.*, 2013)		X		
GEMS (PROCYK; NEUSTAEDTER, 2014)		X		
Tag and Seek (ARKENSON *et al.*, 2014)	X			
Hidden Lion (CHANG *et al.*, 2014)	X			
FreshUP (ZENDER *et al.*, 2014)		X		
Street Art Gangs (ALAVESA; OJALA, 2015)			X	
Gossip at Palace (RUBINO *et al.*, 2015)			X	
Woody (SPIESBERGER *et al.*, 2015)		X		
Pokémon GO (NIANTIC, 2016)	X	X		X
Jurassic World Alive (LUDIA, 2018)	X	X		
Ghostbusters World (FOURTHIRTYTHREE, 2018)	X	X		X
The Walking Dead: Our World (NEXT, 2018)	X	X		

Source: Author

The mapping showed that the majority of LBGs use the *Search-and-Find* and *Follow-the-Path* patterns, which together are present in 76.3% of the analyzed games. *Chase-and-Catch* and *Change-of-Distance* standards are present in 23.7% and 13.1% of LBGs, respectively.

A closer inspection in the *Search-and-Find* and *Follow-the-Path* patterns shows that they share many similarities, especially regarding their implementation and the type of interaction, since both patterns define static locations for players to move to. Conversely, the *Chase-and-Catch* and *Change-of-Distance* patterns rely on dynamic interactions, as the former constantly alters the location being chased and the latter regards only the players' displacement. This unique distinction raises significant impact in the balancing and transposition of these games, and will be further discussed in sections 2.4 and 2.3, respectively.

2.3 Transposition of LBGs

In Chapter 1, transposition is mentioned as one of the key challenges for modern LBGs to become ubiquitous. The concept of transposition was addressed superficially by Macvean *et al.* (2011), that introduced a transposition method called *Location Translation* as part of an authoring tool for the construction of LBGs. This method will be presented in further details in Chapter 3.

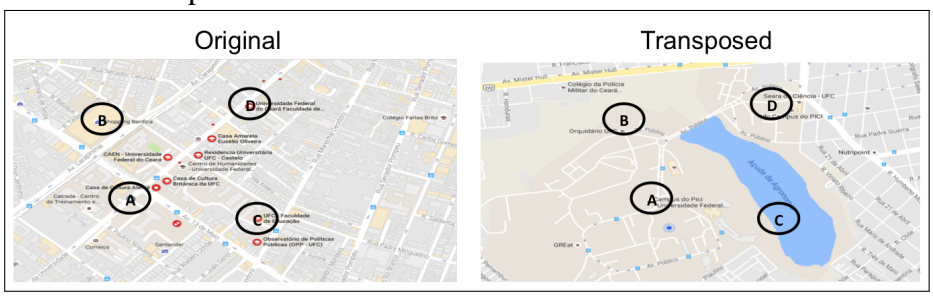
The transposition of an LBG can include multiple features, such as having similar types of POIs in distinct locations, changing the soundtrack according to a particular area, adapting messages displayed to the users, switching non-player character, among others. For instance, a racing game played in Asia could have different music and theme than a transposed counterpart played in Europe. This work focuses on the transposition of maps of LBGs, hence the following definition for transposition is used:

Definition 2.3.1 *The process by which an LBG is replicated and adapted to generate one or more instances of its original version whose differences are limited to the adaptation of the original map to one or more distinct geographic regions.*

The concept exclusively addresses the feasibility of generating new LBG instances that can be executed elsewhere, disregarding any balancing issues. In fact, a plethora of instances may be created, therefore the transposition relates only to the adaptation of games worldwide. Moreover, the transposition of an LBG can restructure multiple aspects and features of the game. For instance, some LBGs can request players to move to specific thematic places, such as touristic places, churches, parks, and others, therefore the transposed version is expected to include similar locations. However, if the game does not establish a direct connection between POIs and virtual places, any site can be used arbitrarily and the transposition affects solely the map of the game.

Transposing an LBG can be difficult due to the differences and the singularity of each place a game can be executed at. For instance, the transposition method has to adapt the game content to multiple contexts, avoid private properties and unreachable areas, and adapt varying interactions implemented by LBGs. A naive transposition of LBGs considering only geographic coordinates is likely to generate bad results (LAATO. *et al.*, 2019). For instance, Figure 8 shows that one of the points (C) has been placed in an unreachable area.

Figure 8 – An example of naive transposition. In this case, the point "C" is positioned in an unreachable area.



Source: Author

Moreover, game developers cannot ensure that the transposed instances of an LBG will always deliver similar gameplay, maintain the game balancing or have minor reconfiguration costs. To better comprehend the many techniques applied, a literature research with LBGs and tools designed to aid in the development and customization of these games was conducted. As a result, in addition to investigating 38 LBGs, nine authoring tools found in the literature were analyzed. From this research, these games and tools were classified into five categories according to the strategy related to the transposition of games. These categories are introduced below and will be detailed in the following sections:

- Reprogramming;
- Displacement-based games;
- Customizable by Authoring Tool;
- Worldwide database of POIs; and
- Methods for automatic transposition.

2.3.1 Reprogramming

Hypothetically, any LBG can be transposed and deployed to different locations. However, for each transposed instance, it is necessary to explicitly reprogram elements such as levels, maps, environment, etc. This task is usually performed without the aid of any software or platform. Hence, LBGs that do not provide these tools are classified as games that demand reprogramming.

Roughly a quarter of the games analyzed in this thesis were developed to be played in a single location and thus required reprogramming to be transposed. It is important to highlight that most of these games have specific goals, such as fostering tourism in a particular region or providing knowledge about a historic site (e.g., *Tag and Seek* (ARKENSON *et al.*, 2014),

Hidden Lion (CHANG *et al.*, 2014), *FreshUP* (ZENDER *et al.*, 2014), and *Gossip at Palace* (RUBINO *et al.*, 2015)).

2.3.2 Displacement-based games

This classification relates only to games that use the *Change-of-Distance* pattern. In this case, the gameplay relies on the displacement of players, regardless of the direction, therefore dismissing transposition. Generally, the *Change-of-Distance* pattern is used in games that stimulate players to engage in healthy applications, such as walking or jogging.

This type of games can make use of different features to gauge the displacement of players. For instance, some games use the GPS of devices to monitor the distance traveled (e.g. *Zombies, Run!* and *BattleSuit Runner*), while others use their accelerometer to estimate displacement based on the number of steps taken (e.g. *The Walk*).

Although games like *Zombies, Run!* (WITKOWSKI, 2013), *The Walk* (START; ALDERMAN, 2016) and *BattleSuit Runner* (SERAPH, 2016) rely solely on the *Change-of-Distance* pattern, other games, such as *Pokémon GO*, *Jurassic World Alive* and *Ghostbusters World* implement this game pattern in some activities or challenges, serving as complement to the overall gameplay.

It is easy to notice that, unlike LBGs with other game patterns, displacement-based games do not require much effort to be executed in multiple locations. In fact, by letting players decide where to move, they do not depend on a specific configuration of locations to run, so games can be played virtually everywhere. Nevertheless, as explained in Section 2.4, these games do not ensure equivalent game balancing between locations. Moreover, by relying exclusively on the *Change-of-Distance* pattern, the gameplay lacks important features available in other games, such as immersion and destinations, since it does not link virtual to real locations.

2.3.3 Customizable by Authoring Tool

A common approach to assist in the transposition of LBGs is the use of authoring tools. These tools are often used in the area of End User Development (EUD) to enable users who are not necessarily developers to modify or extend a software artifact (LIEBERMAN *et al.*, 2006). Therefore, authoring tools for LBGs intend to facilitate the development or reconfiguration of these games by users without programming knowledge.

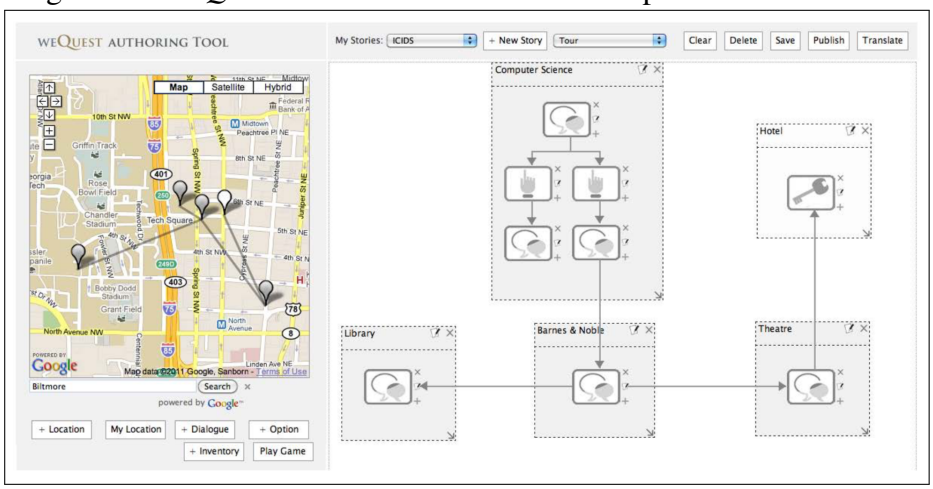
Among many modifications, an authoring tool can assist in the transposition of

LBGs. In this case, the tool can allow users to reconfigure the games so that they can be played in other locations. Approximately half of the LBGs found in the literature make use of this approach, mostly through a web platform (e.g., *Big Game Huntr* (LUND *et al.*, 2010), *See It* (NEUSTAEDTER; JUDGE, 2012), *Tourality* (WORKLINE, 2016), *Exploding Places* (FLINTHAM *et al.*, 2011), *Free All Monsters* (LUND *et al.*, 2011), *Geocaching* (O'HARA, 2008), etc.). Furthermore, games like *PiNiZoRo* (STANLEY *et al.*, 2010) and *GEMS* (PROCYK; NEUSTAEDTER, 2014) allow users to edit their content directly via mobile applications.

The main advantage of using authoring tools to perform the transposition of an LBG is the possibility of customization that it provides to users. In fact, games can be entirely reprogrammed using some of these platforms, therefore making it possible to transpose LBGs to any desired region. However, this approach also presents significant limitations, in particular the need to perform transposition manually. Consequently, it requires the individual reconfiguration of each game instance in multiple places, thus rendering it impractical for worldwide games. As a result, these challenges hinder LBGs that rely on authoring tools from spreading globally.

In this context, it is important to highlight the game *WeQuest* (MACVEAN *et al.*, 2011), which provides an authoring tool that can be combined with a "semi-automatic" method for transposing games. This authoring tool allows users to edit games manually, make adjustments, and correct possible transposition failures, as depicted in Figure 9.

Figure 9 – *WeQuest* allows users to correct transposition failures.



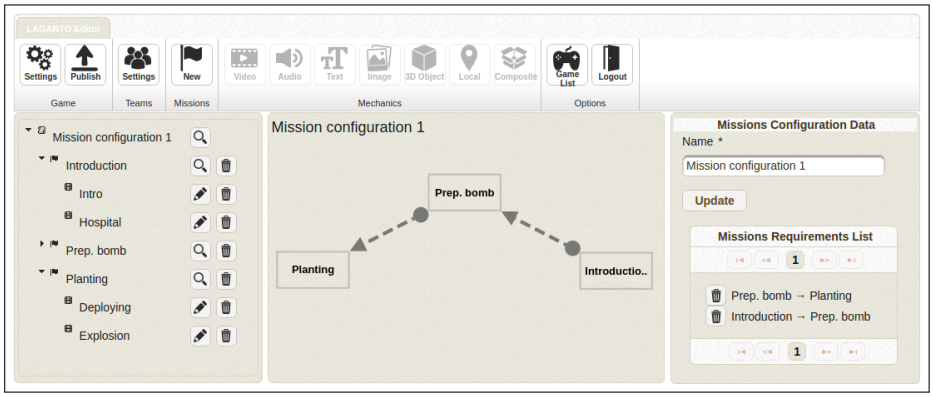
Source: Macvean *et al.* (2011)

In addition, there are several authoring tools for the development of LBGs from scratch (e.g. LAGARTO (NOLETO *et al.*, 2015; SILVA *et al.*, 2017), Aris Games (ARISGAMES, 2014), fAR-Play (GUTIERREZ *et al.*, 2011), ALRA (SANTOS *et al.*, 2013), SILO (WAKE,

2013) and TOTTEM Scout (JURGELIONIS *et al.*, 2013)). The differences between these tools range from the types of game patterns supported, media formats available, platform of execution, and to the support for multiple users.

It is worth mentioning LAGARTO, an authoring tool developed with the support of this thesis' author. The tool supports the editing of game patterns such as *Follow-the-Path*, *Search-and-Find* and *Chase-and-Catch*, and allows game flows to be modelled through dependency lists, as shown in Figure 10. Besides, LAGARTO presents features for the development of *singleplayer* and *multiplayer* games, as well as for the creation of groups of players. The game model used by LAGARTO to represent LBGs inspired the design of the model presented in this work (Section 4.1).

Figure 10 – LAGARTO's main screen depicting a game flow editing.



Source: Noleto *et al.* (2015)

2.3.4 Worldwide database of POIs

A common strategy to allow LBGs to be executed in multiple places is the usage of databases containing POIs. In this case, it is necessary to compile a database containing information about POIs spread throughout the globe. Therefore, players are able to interact with the POIs placed nearby. The main drawback of this approach is the cost and effort to build a database of POIs capable of allowing the game to be played everywhere. As a result, games that use this approach generally suffer from a lack of points available in many parts of the globe. Furthermore, they are likely to provide unbalanced gameplay, as players can be too close or too far away from the POIs, thus leading to unfair competition between players in different areas.

Currently, POIs are present in several location-based services, being common in social networks, geographic databases, navigation systems, etc. In fact, the amount of information about POIs stored in location-based social networks are so massive that recommendation systems

have been developed to better provide this data (YE *et al.*, 2011).

Given the vast number of POIs stored in location-based databases, they are suitable for being used in LBGs. Among the main advantages for using POIs in LBGs are the possibility to select sites with specific features, to access additional information about real places, and to query hotspots from neighborhoods. However, these attributes can vary considerably depending on the database used.

A database of POIs can classify its information into categories, allowing searches for specific locations, like historical sites, libraries, schools, parks, etc., and providing further information that can be used according to the context of each game. This data can be exploited for games with specific purpose, such as tourism and education, or to allow for the correspondence between the visited places in both real and virtual worlds.

The main disadvantages of using databases of POIs for creating LBGs are the risk of accessing outdated information, inconsistency, and restriction of data in some areas. For instance, if a well known POI, such as a library or a mall moves to another location, it may take some time for the database to be updated, meanwhile players may be directed to the old address inadvertently. It is common to encounter obsolete information even in collaborative services due to the dynamics that some points are modified. In addition, there are databases with content exclusive to some countries or regions, which restrict their use.

During research, seven LBGs that are played worldwide using databases of POIs were identified. The games *Ingress* and *Pokémon GO* share the same database, which has been built collaboratively for several years. The games *Parallel Kingdom* (PATRO *et al.*, 2012), *Jurassic World Alive*, *The Walking Dead: Our World*, and *Ghostbusters World* make use of POIs and maps from *Google Maps*. Finally, the game *Woody* uses the database *Treepedia*, that is an online catalog containing information about trees in some countries.

Nevertheless, despite having information about places in many countries, the database of POIs used by *Ingress* and *Pokémon GO*, does not contain the necessary amount of data for the games to run everywhere. In fact, there are several countries and cities where you cannot play these games [1]. Recently, a survey conducted with 2612 *Pokémon GO* players revealed that 27.3% of participants claimed to have stopped playing due to issues such as technical problems and the lack of content in the area. In this case, the respondents criticized the unequal gaming possibilities due to the POIs being concentrated in city centers (ALHA *et al.*, 2019).

[1] http://www.gamespot.com/articles/pokemon-go-players-in-rural-areas-upset-over-lack-/1100-6441696/

In addition, this database also does not provide equity between the geographical distribution of POIs, hence culminating in considerable difference in gameplay between neighborhoods, cities, and regions. This issue has caused many impacts on game balancing, thus rendering the games unfair to many players[2].

As discussed, the number and distribution of POIs in a database are key for enabling games to be played in multiple locations, therefore a large worldwide database of POIs is required. Recently, *Google* has released an API for developers of LBGs to build their games on top of the *Google Maps* database[3]. In fact, some games released in 2018, like *Jurassic World Alive*, *The Walking Dead: Our World* and *Ghostbusters World*, were developed to take advantage of this database, hence they can be run in many places.

The approach used by these games does not feature a transposition in practice. In this case, instead of transposing the game according to the players' location, they used the database of POIs to create a game map that comprised the globe. Consequently, the game is a unique instance in which players navigate according to their location in the world.

2.3.5 Methods for automatic transposition

This classification relates to works that use automatized methods to conduct the transposition of any LBGs' features. The purpose is to avoid or alleviate the need to perform manual configuration of game instances. In some cases, the methods partially require human intervention, thus being considered semi-automatic.

There are a few researches devoted to the use of automatic transposition. Among the games and authoring tools analyzed, only *WeQuest* (MACVEAN *et al.*, 2011) and *Easter Egg Hunt* (JORDAN *et al.*, 2013) benefited from methods of automatic or semi-automatic transposition. However, in both games there are particularities and drawbacks to be considered. These works share similarities to the this research, and are detailed in the next chapter.

2.4 Game Balancing

Game balancing can also be understood as the difficulty level of a game, and is one of the fundamental aspects for the development of commercial games (OLESEN *et al.*, 2008). Theoretical psychology studies conducted with players suggest that the appropriate level of

[2] http://www.miamiherald.com/news/nation-world/national/article89562297.html
[3] https://cloud.google.com/maps-platform/gaming/

challenge is a key factor for a satisfactory gaming experience (OLESEN *et al.*, 2008; KOSTER, 2005).

According to Csikszentmihalyi (1990), the contribution of balancing to a pleasant gaming experience is also derived from the so-called *Flow Theory*, since the difficulty level of a game when compatible with the skills of the player composes one of the nine factors of that theory (CSIKSZENTMIHALYI, 2000). In general, very difficult games are frustrating, whereas very easy games lead to boredom. This relationship can be seen in Figure 2, whose central area of the graph represents the desired balancing, which is an essential condition for players to remain interested in the games from beginning to end (ANDRADE *et al.*, 2006).

For being recognized by the game developer community as one of the main factors in the commercial success of a game, several efforts have been devoted to the study and development of techniques for balancing digital games. Currently, game balancing is generally achieved in two distinct ways. Either by using predefined settings (e.g. beginner, intermediate and advanced) that allow users to choose the difficulty level they wish to play, or by artificial intelligence techniques that may even use dynamic balancing (ANDRADE *et al.*, 2006; OLESEN *et al.*, 2008).

Usually, adjusting game balancing is one of the most difficult and time-consuming tasks in game development due to the repeated cycles of tweaking and testing, which aims to improve features such as depth, pitch, fairness, randomness, and variety (JAFFE *et al.*, 2012). Besides, Jaffe *et al.* (2012) argue that more complex games are also more difficult to balance, since small adjustments can have unexpected consequences in other areas of the game.

A key point related to balancing is the concept of fairness between players. For example, even in traditional games like Chess, it is argued that the beginning of the matches is unfair because the white pieces have the advantage of executing the first move. In this case, one can evaluate the fairness of the game based on the winning rate between players with white and black pieces (JAFFE *et al.*, 2012). Obviously, fairness in digital games may involve multiple elements, ranging from the initial position of a character in the virtual world, the availability of items players can collect, and even the random generation of elements and challenges of a game. Therefore, it is important to consider these factors so that the digital games remain balanced and consequently fair.

In the case of LBGs, on top of the factors mentioned above, the location of POIs and the difficulty to move between places add extra complexity to the game balancing. Actually, since the gameplay relies on the movement of players in the real world, it is presumable that

the game balancing in LBGs is mostly determined by the selection of suitable destinations for players to move to. Therefore, it is especially complex to balance them due to the inherent link between virtual and real worlds. This task may depend on a multitude of factors and settings that can be combined to form countless options. For instance, distance, time, topology, transport availability, weather, and public safety are just a few examples of real world features that can influence the gameplay, and hence game balancing in LBGs.

Besides, players can start game sessions virtually everywhere, thus each execution can contain a different game configuration that needs to be balanced. The myriad of possibilities and configurations makes manual game balancing an impractical job. Thus, an automatic approach to improve game balancing in LBGs is needed.

Many works have been developed to foster the use of automatic game balancing method in multiple game genre, such as puzzles (ASHLOCK, 2010), real-time strategy games (OLESEN *et al.*, 2008), serious games (WESTRA *et al.*, 2008), fighting games (ANDRADE *et al.*, 2005), among others. However, LBGs have not seen many studies in this field, and the few works that mention it do not specifically address the issue (JACOB; COELHO, 2011; MACVEAN *et al.*, 2011). Therefore, most LBGs deliver unbalanced and unfair experiences depending on the area they are being played.

2.5 Balanced Transposition of LBGs

Sections 2.3 and 2.4 presented the concepts and challenges behind the transposition and game balancing in LBGs. This section discusses how these definitions can be combined to create what is called *Balanced Transposition*, a new concept that can be explained as:

Definition 2.5.1 *A variation of the transposition process that focuses on preserving the game balancing of the original game when generating transposed game instances.*

An essential aspect of the Balanced Transposition of LBGs is the examination of locations to which games will be migrated. Such analysis must consider geographical differences between regions due to their influence to game balancing. This analysis is fundamental to enable the transposition of LBGs in such a way that it preserves their balancing in multiple locations.

Conducting the transposition of an LBG is not a simple task because distinct game patterns may be used, inhospitable or forbidden areas should be avoided, and there can be a vast number of attributes to be analyzed depending on the game and the place of execution. Besides,

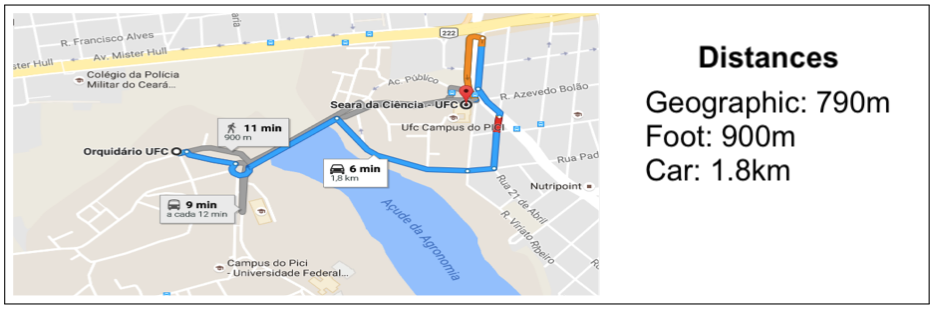
to preserve the game balancing across instances in distinct regions, topographical peculiarities, relief and means of transportation must be taken into account.

In general, the *Follow-the-Path* and *Search-and-Find* game patterns present similar challenges related to transposition, since both depend on the relocation of specific geographic coordinates. However, a direct transposition of the geographical coordinates from one place to another tends to produce poor results, as showcased by Figure 8.

Another issue that must be highlighted relates to security, since LBGs are expected to be executed everywhere. It is important that the locations selected to compose the game are not placed in regions deemed dangerous. This and other characteristics mentioned previously illustrate the challenges addressed by this work.

With respect to balancing, using the absolute distance between the points is oblivious, because the actual displacement to be performed by the players can be significantly different, according to the destination or means of transportation. For instance, Figure 11 presents a comparison of the possible routes between two points considering the absolute distance (also known as geographic distance), on foot and by car. Therefore, one can conclude that a transposed instance of an LBG can present locations with equal absolute distance in comparison to the original game, however, only these factors do not ensure that their game balancing is equivalent.

Figure 11 – Comparison between paths connecting two points considering geographic, walking and car distances.

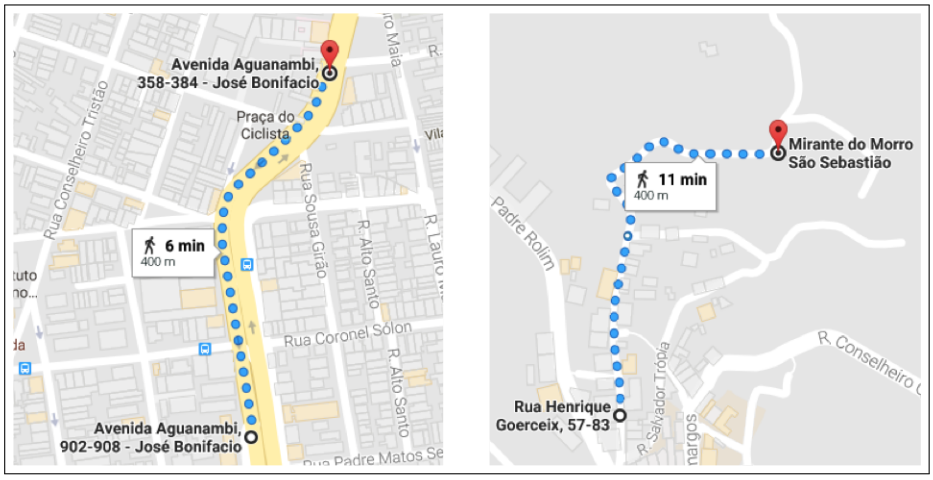


Source: Google Maps

In the case of *Chase-and-Catch* and *Change-of-Distance* patterns, a superficial analysis tends to indicate that the balanced transposition of LBGs that use these patterns is easier to perform, as they do not make use of specific destinations. In fact, games that use these game patterns tend to be played in different locations with little or no changes. However, it is not possible to ensure that game balancing is equivalent between instances of the same game when they are executed at different locations due to differences in the topography of these regions. For instance, consider the game “Tag” being played by individuals in distinct regions, a

player moving through a flat area would have advantages in terms of speed and distance traveled compared to an opponent playing in a mountainous location. Other traits to be considered are seasons and climate, as distinct locations may also have disparate weather conditions that can potentially affect gameplay. In this case, it is clear that considering only the displacements to define game balancing regardless of the peculiarities of each region leads to unbalanced games. Figure 12 shows that paths with the same distance can present significant difference in time depending on the location.

Figure 12 – Comparison involving the time to walk between two sites in regions with different topography.



Source: Google Maps

2.6 Procedural Content Generation

Complex games require teams composed of hundreds of skilled personnel just to create game content (HENDRIKX *et al.*, 2013). Hence, relying solely on humans to produce the vast amount of virtual content present in modern digital games makes production a slow, costly and risky process. Moreover, the growth in the player population and the lack of scalability in the human production pipeline indicates manual content production is not sustainable (IOSUP, 2011). Furthermore, the previously mentioned scenario combined with the ever decreasing price of computers has led to the development of automated techniques to improve the creation of game content, thus fostering studies about PCG.

PCG is an emerging field in the game industry that has recently received more attention from academy and industry due to the increasing demand for high-quality games and the growing costs incurred from producing better titles. It is defined by Togelius *et al.* (2011) as

the algorithmic creation of game content with limited or indirect user input.

Nowadays, PCG is used to produce multiple types of content, such as textures, sound, maps, buildings, vegetation, levels, meshes, terrain, puzzles, etc. Since there are varying content to be produced, the methods employed in PCG are also diverse, including Pseudo-Random Number Generation, Generative Grammars, Image Filtering, Spatial Algorithms, Modeling and Simulation of Complex Systems, and Artificial Intelligence (HENDRIKX *et al.*, 2013).

This work proposes the use of PCG as the foundation for the transposition of LBGs maps. The proposed method consists in devising a generic model containing key aspects of an LBG and using algorithms to automatically generate new maps of the game based on the game model and according to specific locations. In turn, this approach operates as a transposition method since it allows for an LBG designed in a particular location to be mapped and played elsewhere. Moreover, the method is responsible for generating a map that resembles the original game as much as possible, therefore PCG must consider features such as gameplay, balancing, correctness, etc. Chapter 4 details the proposed game model, resources and algorithms used to conceive a robust PCG approach for transposing LBGs.

The first works to use PCG to generate game maps date back to the early 2000s. For instance, Parish and Müller (2001) developed *CityEngine*, a software based on L-systems to model cities. The solution generates maps of highways and streets from multiple image maps such as land-water boundaries and population density. Using L-systems to create these maps provides good support for branching and has the advantage of database amplification (SMITH, 1984).

Moreover, Sun *et al.* (2002) used 2D images as input maps and a rule-based system to generate maps of roads. The proposed method created a virtual traffic network based on image-derived templates and a rule-based generating system. A key feature of this approach is the ability to generate roads while avoiding illegal areas, and connect dead-end roads to form a network. Although the method does not consider traffic flow and topography, it is suitable for generating roads for games and virtual cities.

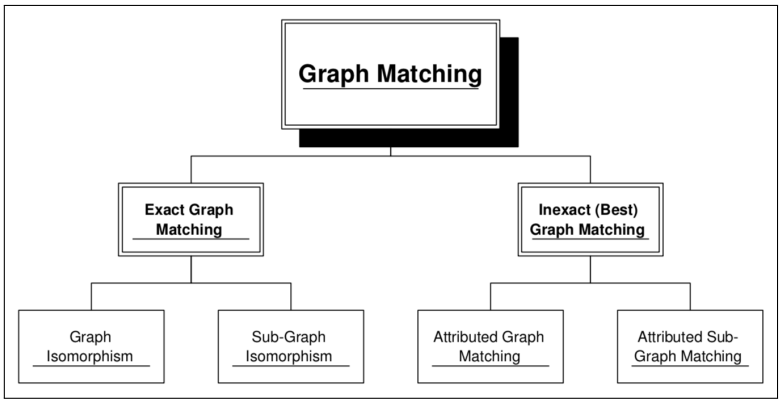
2.7 Graph Matching Problem

Section 2.6 mentions that the proposed approach relies on a game model that represents LBGs. This model works as a generic representation for distinct LBGs that includes only the data required for the transposition, thus allowing the approach to work with several types of

games. In this case, the game model can be defined as a weighted graph $G = (V, E, W)$, where V is a set of nodes representing the places to visit, E is a set of edges ($E \subset V \times V$) symbolizing an existing path between places, and W is the set of edge weights ($W : e_{i,j} \rightarrow \mathbb{N}+$), representing the cost to move between places (Chapter 4 presents the game model in further details). Consequently, transposing the map of an LBG to a new region consists in finding suitable locations that will constitute a similar graph representation, meaning node, edges and weights are as similar as possible in both original and transposed versions.

The challenge of finding graphs that are similar is known as *Graph Matching Problem* or *Graph Isomorphism Problem*, and is in NP, neither known to be in P nor NP-complete (LIVI; RIZZI, 2013). This is a classic challenge in Computer Science that has numerous variations, encompassing weighted/unweighted, directed/undirected, and cyclic/acyclic graphs (BENGOETXEA, 2002). The GMP has applications in fields such as Molecular Biology (AMIN *et al.*, 2010), Chemistry (AKUTSU; NAGAMOCHI, 2013), Computer Vision (LI; WACHS, 2012), Bioinformatics (LAJEVARDI *et al.*, 2013), Robotics (CORTES *et al.*, 2006), etc.

Figure 13 – Graph matching main classification: Exact Graph Matching and Inexact Graph Matching.



Source: Bengoetxea (2002)

The GMP can be classified into two main types, known as *Exact Graph Matching* and *Inexact Graph Matching*, with additional subcategories associated to each type (BENGOETXEA, 2002), as shown in Figure 13. The *Exact Graph Matching* consists in, given two graphs $T = (V_t, E_t)$ and $S = (V_s, E_s)$, finding a bijective mapping $f : V_t \rightarrow V_s$ such that for each $(V_{t_i}, V_{t_j}) \in E_t$ there is a $(f(V_{t_i}), f(V_{t_j})) \in E_s$. If $|V_t| = |V_s|$ the challenge is called *Graph Isomorphism Problem*, otherwise if $|V_t| < |V_s|$ it is called *Subgraph Isomorphism Problem*, as the resulting graph $R = (V_r, E_r)$ consists in a subgraph of S, where $V_r \subset V_s$ and $E_r \subset E_s$.

Conversely, *Inexact Graph Matching* (also known as *Homomorphic Graph Matching*)

indicates that an isomorphism between two graphs can not be extracted. Therefore, the number of matching vertices V and edges E can differ, meaning that the challenge is to find the best matching between them. As a result, these methods search for a non-bijective mapping $f : V_t \rightarrow V_s$ between two graphs $T = (V_t, E_t)$ and $S = (V_s, E_s)$.

This work focuses on a particular case of *Subgraph Isomorphism* that includes weighted graphs (WGMP), a problem that is known to be NP-complete (COOK; HOLDER, 2006). In summary, the challenge consists in, given a target graph $T = (V_t, E_t, W_t)$ and graph to be searched $S = (V_s, E_s, W_s)$, finding a subgraph $R = (V_r, E_r, W_r)$, where $V_r \subset V_s$, $E_r \subset E_s$, and $W_r \subset W_s$, that better resembles T. In this case, it is required to find a resulting subgraph R that is isomorphic to T, as there must exist a bijective mapping $f : V_t \rightarrow V_r$. In this case, the challenge is to explore a larger graph (search space) looking for subgraph that best resembles a particular graph (game model). Obviously, since LBGs are represented as weighted graphs, the approach must consider both connectivity and weights as similarity features. Furthermore, in this work, the best possible R must satisfy a fitness function that measures the similarity between R and T by comparing the difference in weights between edges. Chapter 4 details the process that allows for LBGs to be represented by weighted graphs, as well as presents the function used to optimize the similarity between graphs T and R.

2.8 Algorithms

This work presents three distinct algorithms to tackle the transposition of maps of LBGs as a WGMP. Hence, this section presents the algorithms that were used as the basis in the development of this research.

Section 2.8.1 details the Ullmann's Algorithm, that was adapted to include the ability to process weighted graphs, Section 2.8.2 presents the concepts and steps that make up the MCTS, and Section 2.8.3 presents a discussion about the use of Genetic Algorithms to tackle the WGMP.

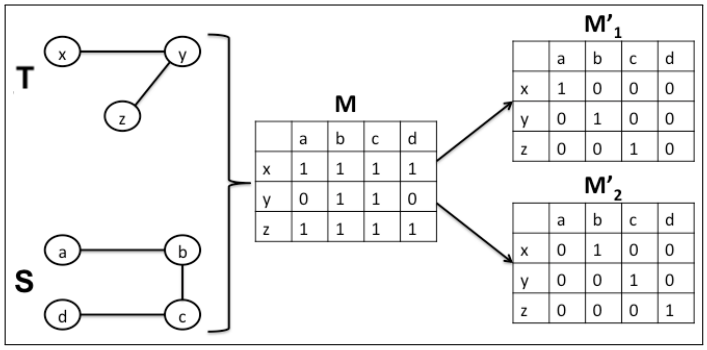
2.8.1 Ullmann's Algorithm

One of the most successful methods for graph and subgraph isomorphism is the seminal algorithm proposed by Ullmann in 1976 (ULLMANN, 1976). The approach relies on a depth first tree search enhanced with a refinement procedure that classifies vertices according

to their connectivity to narrow down the search space. This work presents an adaptation of the Ullmann's algorithm that can cope with weighted graphs.

Furthermore, the Ullmann's algorithm consists in a series of independent explorations that are suitable to parallelism. Hence, the proposed method makes use of this trait to improve runtime performance.

Figure 14 – The root matrix M generates matrices (M') that encode candidate graphs.



Source: Author

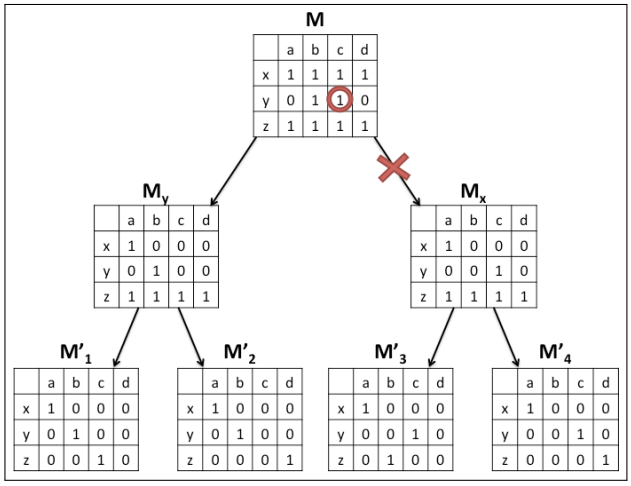
Ullmann's algorithm takes as input a graph T (target) and a graph S (search space), and seeks a graph R ($R \subset S$) that is isomorphic to T. The method represents graph isomorphism using a matrix M' with dimensions $|v_T| \times |v_S|$, where each line contains exactly one "1" and each column contains at most one "1". M' is generated from a root matrix M whose elements m_{ij} are defined as follows (Figure 14):

$$m_{ij} = \begin{cases} 1, \text{ if the degree of } v_j \in S \text{ is greater than or equal to the degree of } v_i \in T \\ 0, \text{ otherwise} \end{cases} \tag{2.1}$$

Moreover, the algorithm prunes the search space by examining the connectivity between vertices and their neighbors. By definition, two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if each of their vertices has a one-to-one mapping that preserves adjacency ($f : V_1 \rightarrow V_2$ such that $u, v \subset E_1$ if and only if $f(u), f(v) \subset E_2$). Consequently, if a vertex $v_i \subset V_1$ or any of its neighbors does not match a vertex $v_j \subset V_2$ and its neighbors, all solutions containing this mapping can be eliminated. Hence, it is possible to safely set $m_{ij} = 0$ in matrix M. Figure 15 exemplifies how this process prunes the search space.

In summary, depending on the connectivity of the graphs, Ullmann's algorithm can perform big prunes to the search tree. Conversely, if the search space is a complete graph no pruning occurs and the algorithm processes all possible cases, hence operating as a brute-force approach.

Figure 15 – An example of refinement performed by Ullmann's algorithm.



Source: Author

2.8.2 MCTS

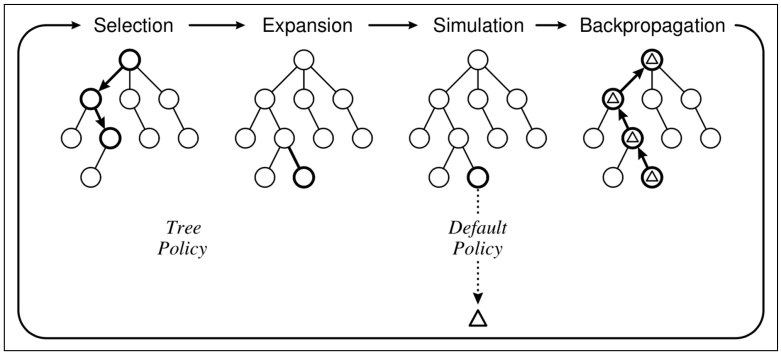
MCTS is best known for its ability to compete at an expert level in the game *Go* (COULOM, 2007). However, MCTS has successfully been applied to multiple areas, such as optimization, real-time strategy games, general game playing, and complex real-world planning (BROWNE *et al.*, 2012).

Additionally, MCTS has also been used to find near optimal solutions to large state-space Markovian Decision Problems (KOCSIS; SZEPESVáRI, 2006). Since WGMP can have graphs with varying sizes depending on the games, the search space can grow exponentially, therefore making deterministic approaches unfit to the challenge. Consequently, non-deterministic methods can be used in these cases for their capacity to deliver good results under a predefined computational budget.

MCTS is used in this research due to its ability to optimize the exploration of the search space using information collected on previous searches. During execution, MCTS probes the search space and builds a partial tree that is used to assess each solution. Therefore, the algorithm can focus on branches of the search space that deliver more promising results, while avoiding branches that produce bad outcomes.

The algorithm is divided into four main tasks, as shown in Figure 16. In the first step (selection), the method uses a tree policy to select a branch of the tree based on the values of tree nodes. Second (expansion), one or more child node is added to expand the tree and build a solution that will be evaluated. In the third step (simulation), a simulation with this solution is executed according to a default policy and results are evaluated. Finally, in the last

Figure 16 – General steps that compose an MCTS algorithm.



Source: (BROWNE *et al.*, 2012)

step (backpropagation), the results from the simulation are used to update the values of nodes, thus guiding further selections.

2.8.3 Genetic Algorithms

Evolutionary algorithms, such as GAs, Particle Swarm Optimization and Simulated Annealing, have been widely applied to numerous NP-hard problems, such as the Traveling Salesman Problem (TSP) (POTVIN, 1996) and the GMP (KRCMAR; DHAWAN, 1994; XIU-TANG; KAI, 2008). These algorithms are known for finding sub-optimal solutions in polynomial time, thus being suitable for problems that have large search spaces. This work uses a GA to address the WGMP.

The decision to use a GA instead of any other evolutionary method is based on the work developed by Li *et al.* (2016), that has analyzed the implementation of three heuristic optimization algorithms: Simulated Annealing, (1+1) evolutionary algorithm, and GA to the subgraph isomorphism problem. They concluded that GA presented better results than the other algorithms in most cases.

GAs focus on evolving the most promising solutions using nature-based operations, such as *Crossover*, *Natural Selection* and *Mutation*. Consequently, the algorithm does not have to explore the entire search space, and keeps a balance between execution time and the quality of the solutions.

2.9 Conclusion

In this chapter, the concepts and challenges for the understanding of transposition and game balancing of LBGs were discussed. In the context of the present work, the definitions

of Ubiquitous Computing, Pervasive Computing, Ubiquitous Games, Pervasive Games and, mainly, Location Based Games were clarified. In addition, the game patterns found in LBGs according to Lehmann (2012) were presented, and a mapping of these patterns in modern LBGs was performed.

The challenges to transpose these games considering distinct game patterns were also discussed, and the main issues to generate balanced instances of LBGs were presented. Moreover, it was shown how the proposed solution relates PCG methods and discussed how the transposition of LBGs can be addressed as a *Subgraph Matching Problem*. Lastly, the methods used as the foundation to the transposition algorithms were introduced.

3 RELATED WORK

This chapter presents related methods and techniques adopted by both the Industry and the Academy involving PCG, Subgraph Matching algorithms, transposition and balancing of LBGs. The chapter is divided into five sections detailed as follows. Section 3.1 presents the works developed to automatize the transposition of LBGs to multiple locations. Section 3.2 discusses methods and approaches related to game balancing. Section 3.3 shows how PCG is applied to the generation of maps and game levels, and Section 3.4 details graph and subgraph isomorphism algorithms that are related to this research. Finally, Section 3.5 concludes this chapter.

3.1 Methods for automatic transposition

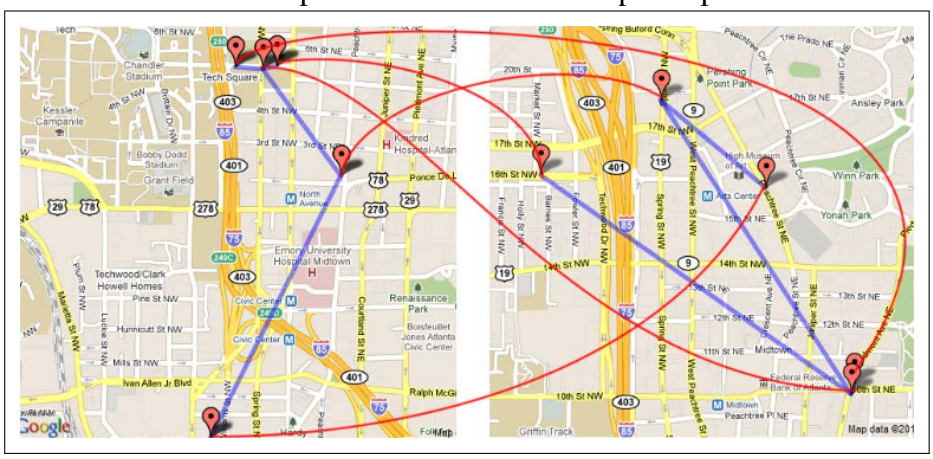
Transposition is a key feature for the success and popularization of an LBG. As discussed in the previous chapter, using the player's location as a critical element of gameplay implies great consequences and challenges for the development of these games.

This research focuses on the automatic transposition of LBGs' maps, however, there are a few works that have proposed similar methods. Among the 38 games and authoring tools analyzed in Chapter 2, only *WeQuest* (MACVEAN *et al.*, 2011) and *Easter Egg Hunt* (JORDAN *et al.*, 2013) used transposition methods that can be classified as automatic or semi-automatic.

WeQuest is a game created in conjunction with a web authoring tool that allows for the creation and editing of geo-referenced alternate reality games. LBGs developed with this tool can be downloaded from the Internet and executed in a mobile application. This platform allows users to model games that implement exclusively the *Follow-the-Path* pattern using a directed acyclic graph. In this case, nodes of the graph are associated with specific locations via geographic coordinates, and the graph's edges designate the path that players must follow when playing the game.

To complete the transposition of a game using the *WeQuest*'s authoring tool, it is necessary to appropriately reposition each node of the graph to a corresponding new location. This task is lengthy and time-consuming depending on the number of sites to be visited. So, the authors proposed an algorithm called *Location Translation* to facilitate this operation. Given that the nodes of the graph represent locations in an LBG, the algorithm is responsible for mapping each node to a site in regions where players wish to execute the game (Figure 17).

Figure 17 – Depiction of the *Location Translation* algorithm. The blue lines represent the edges of the graph and the red lines depict the correspondences between transposed points.



Source: Macvean *et al.* (2011)

Since the *WeQuest* platform was created to design alternate reality games, whose points of interaction in the virtual world must relate to the ones in the real world. The *Location Translation* algorithm was developed with the purpose of selecting virtual places with the greatest possible similarity to the real locations. In this way, if a school was selected as a site in the original game, its corresponding node in the transposed game tends to also be a school. After classifying eligible points by similarity, the algorithm aims at minimizing the difference in distance between original and transposed areas, thus trying to maintain a minimum balance between original and transposed games. However, the algorithm does not assure or evaluate the balancing of the transposed games.

Location Translation is a dynamic programming algorithm that aims to minimize a cost function related to the difference of similarity and distance between points. The method works by querying a set of candidate locations from the *Google Maps API*, but users can make revisions and edit selected points since the method is deemed semi-automatic.

In general, the algorithm satisfies the needs of the platform and the types of games developed in it. However, several disadvantages prevent the use of *Location Translation* for other LBGs. First, the proposed algorithm only works with games that implement the *Follow-the-Path* patterns. Moreover, the method treats game balancing between instances as a secondary property. Lastly, *Location Translation* presents a high complexity and high memory consumption for being a dynamic programming algorithm ($O(n_{max} * |L|)$, where n is the number of candidate locations in the new region and L is the number of nodes in the original graph). In summary, the algorithm is unsuitable depending on the game (number of nodes) and the location (amount of candidate

sites) the transposition is taking place.

On top of that, the game *Easter Egg Hunt* was developed to identify POIs based on users' displacement. To achieve this goal, the game implements the *Search-and-Find* pattern through a semi-automatic distribution of virtual items (easter eggs) in a specific area (Figure 18). This approach is not completely automatic because it requires manual configuration about the region the game is taking place, including exclusion areas such as lakes, private properties, etc.

Figure 18 – In the game *Easter Egg Hunt*, points (in red and blue) are automatically distributed.

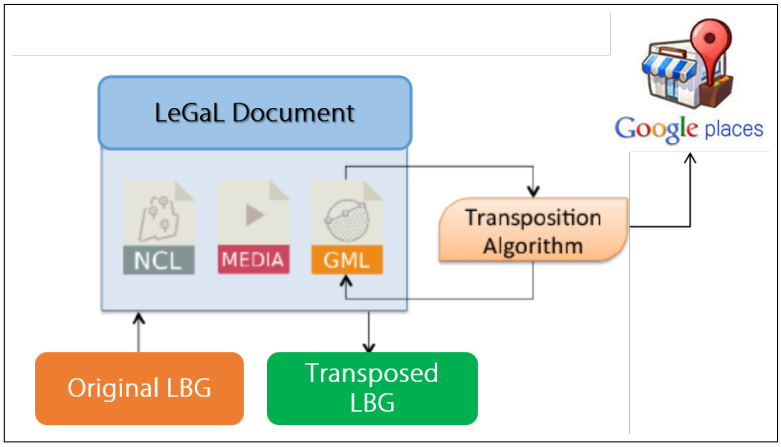
Source: Jordan *et al.* (2013)

The main disadvantages of the method used in *Easter Egg Hunt* are the need for manual adjustments of the area to which the game will be transposed, the exclusive support for the *Search-and-Find* pattern, and the disregard for the game balancing. In fact, games generated using this approach are played only in the area of the transposition, regardless of routes, viability of access, relief, among others.

In addition, this work proposes a fully featured game model that allows for the design of LBGs (FERREIRA *et al.*, 2017). This game model focuses on the development of mission-based games using spatial and temporal relationships between game elements. To showcase the potential of their game model, the balancing algorithm presented in (MAIA *et al.*, 2017) was adapted to transpose a game called *AudioRio* from Fortaleza to Curitiba, in Brazil. Figure 19 shows how the game model can make use of a transposition algorithm seamlessly.

Later, I have developed a method capable of transposing maps of LBGs while

Figure 19 – Transposition of maps using the game model presented by (FERREIRA *et al.*, 2017).



Source: Ferreira *et al.* (2017)

preserving game balancing (FERREIRA *et al.*, 2019). The approach builds on previous researches developed by the author of this work in (MAIA *et al.*, 2017) and (FERREIRA *et al.*, 2017) to transpose maps of LBGs while focusing on maintaining game balancing and gameplay. The transposition algorithm uses MCTS, an approach widely known for its potential as an AI technique capable of mastering the game *Go* (COULOM, 2007). In this case, the method takes POIs from a previously defined LBG as input and uses MCTS to search for a set of POIs in a new location that best matches the original game.

In (FERREIRA *et al.*, 2019), tests were conducted to transpose a game called *Quest for the Cathedral*. The game was originally designed to be played in Fortaleza, and a transposed version was generated in Amsterdam, Netherlands. The steps taken by the approach are as follows. First, the algorithm gathered information from the Google Maps API to compile a set of candidate POIs C (in Amsterdam) to compose the transposed version. Next, it uses MCTS to explore the search-space looking for a subset of POIs R ($R \in C$) that minimize map balancing differences, thus making the transposed version similar to the original game. This approach was evaluated and the results indicate the algorithm can help deploying LBGs to multiple places (FERREIRA *et al.*, 2019).

3.2 Game Balancing

As previously mentioned, balancing games is a difficult and time-consuming task that requires extensive testing and calibration. Besides, game balancing presumably gets harder with more complex games, since slight modification can potentially impact other game features

(JAFFE *et al.*, 2012). As a result, researchers have focused on the design of automatic and dynamic game balancing techniques, ranging from the use of artificial neural networks (OLESEN *et al.*, 2008) to reinforcement learning techniques (ANDRADE *et al.*, 2005).

Recently, I have defined two metrics to gauge aspects of game balancing in LBGs (MAIA *et al.*, 2017). The first, called Internal Difficulty Level ($\mathfrak{I}$), focuses on examining the internal balancing, i.e., it evaluates whether POIs are distributed evenly and measures the effort for players to move to the closest POIs. The second, called Minimum Balancing Difference ($\mathfrak{M}$), compares two instances of the same game and assesses dissimilarities in their game balancing. It establishes a direct comparison between distinct LBGs or distinct regions of the same game. $\mathfrak{M}$ uses the concept of graph similarity and graph matching distance, which has been investigated in many works (XU *et al.*, 2013; SANFELIU; FU, 1983; RAVEAUX *et al.*, 2010). These measurements will be detailed in Chapter 4 as they are key elements in this thesis.

Furthermore, an automatic method that uses MCTS and *Google Maps* to improve game balancing in LBGs was presented by the author of this work (MAIA *et al.*, 2017). The approach builds on MCTS to adjust the game balancing on demand -according to the player's location- and to cope with games that have varying amounts of POIs. In this case, MCTS was used to optimize exploration using information collected on previous searches, hence focusing on promising solutions while devoting minor efforts to portions of the search space that generate bad outcomes. The authors have showed the efficiency of the approach when applied to the game *Pokémon GO*.

3.3 Procedural Content Generation

PCG has been used in games to generate a wide variety of content, ranging from textures to game rules. This work uses PCG to transpose maps of LBGs to multiple locations. As mentioned in Section 3.2, maps are a key component in LBGs as they establish a link between real and virtual worlds, thus acting as game levels and being vital to provide a satisfying gameplay. Similarly, many games have their gameplay based on the player controlling an agent in a virtual space, therefore these spaces can also constitute game levels. Consequently, works that use PCG to address the challenge of creating game maps and levels were investigated.

3.3.1 Game Maps

This section presents approaches that use automatic or semi-automatic methods to create maps for games. Nevertheless, the works investigated here include features related to the transposition of maps of LBGs, such as paths, roads, traffic, etc. As a result, researches related to terrain generation that focus on height maps, vegetation, relief, and aesthetic purposes are not discussed.

Glass *et al.* (2006) analyzed road maps of informal settlements in South Africa using aerial photography to determine procedural techniques capable of replicating them. As a result, the work unveiled that a combination of Voronoi diagrams and subdivision provides the closest match to informal settlements, while a combination of L-systems, Voronoi diagrams and subdivision creates the closest pattern to a structured informal settlement. Therefore, new maps can be recreated from this combination of techniques and parameters.

Patel (2016) released a method for generating polygonal maps that uses a graph structure to model features linked to gameplay constrains, such as elevation, roads, river flow, quest locations, among others. The approach uses Voronoi polygons generated from a set of random points to create an initial mesh and its corresponding graph, then it benefits from the existing duality between Voronoi Diagrams and Delaunay Triangulations to extract a second graph depicting adjacency relations. Next, properties such as border, terrain, and lakes are linked to the graph representing Voronoi corners and features such as elevation, paths and quests are associated with the graph storing Delaunay edges. Finally, the mesh is subdivided and noise is applied to create a more detailed and smooth map.

Abuzuraiq (2017) builds on (PATEL, 2016) to create an algorithm to generate maps that can be used in the distribution of terrain and converting or linking Mission Graphs to game spaces. The method takes a planar graph G as input and solves the problem of partitioning a planar graph using the A* search algorithm coupled with a heuristic that creates quotient graphs isomorphic to a constraint graph C. To address the challenge of coping with increased search spaces (G is large), the algorithm uses a coarsening step that partitions G into a new graph G' that is smaller in size but does not over-limits the search space.

A work for generating maps to Real-Time Strategy (RTS) games using a search-based method was presented by Togelius *et al.* (2010). In RTS games, maps are crucial to gameplay as they also constitute the game levels, likewise many LBGs. The proposed approach generates maps for the RTS game *Star-Craft* using a multiobjective evolutionary algorithm. The

method relies on a set of fitness functions related to attributes like playability, fairness, skill differentiation and interestingness. To validate the work, a simulation of a character moving between two points along the fastest possible path was implemented using the classical A* algorithm. In this case, distinct fitness measures (mainly related to distance) were proposed to reflect game characteristics.

Dormans (2010) tackles the challenge of generating game maps associated with missions in action-adventure games, such as *The Legend of Zelda: The Twilight Princess*. These games have their gameplay founded on enjoyable exploration, flow and narrative structure, attributes that rely on game maps containing a set of missions, similarly to most LBGs. The method considers missions and spaces as two separate structures generated independently. It uses generative grammars to first create a graph representing missions and then generate spaces to accommodate these missions. This approach requires a collection of rules to output entertaining and diverse game maps.

3.3.2 Game Levels

There are many works devoted to the generation of game levels, mostly related to 2d platform games. For instance, Sorenson and Pasquier (2010) present a generative system for the automatic creation of video game levels using the FI-2Pop genetic algorithm, Snodgrass and Ontañón (2014) and Snodgrass and Ontañón (2017) use Markov chains calibrated using to a series of 2d maps of game levels designed by humans.

Compton and Mateas (2006) build game levels by repeating and reshuffling a few game components according to rhythmic actions thus making players experience better game "flow", Smith *et al.* (2009) and Moghadam and Rafsanjani (2017) use a grammar-based method to also generate 2d platform levels based on rhythms, and Pedersen *et al.* (2010) generates *Mario Bros* levels randomly by traversing a fixed width and placing gaps, blocks and enemies following some heuristics.

A procedural level generator based on a GA that works for any game or content type was present in Adrian and Cosío (2013). Their approach uses a fitness function that calculates the difference between a desired difficulty curve and the difficulty curve calculated from the candidate content, thus levels are created to best fit the desired curve.

Lelis *et al.* (2018) and Reis *et al.* (2015) used semi-automatic level generators that combines a number of annotated segments into a full-sized level of the game. These segments

were evaluated by humans with respect to their perceived enjoyment, aesthetics, and difficulty, therefore the resulting levels were deemed more enjoyable and visually pleasing.

Smith *et al.* (2018) present an approach that models acyclic dungeon levels as graphs to satisfy gameplay and design constraints. The work models constraints and graphs as an Answer Set Programming (ASP) problem to produce dungeon levels that are validated using a domain-independent solver.

3.4 Weighted Graph Matching Problem

Section 2.7 mentions that an LBG can be modeled using a weighted graph, and that the transposition can be depicted as a particular case of *Graph Matching* called WGMP. In this case, the challenge is to explore a larger graph (search space) looking for subgraph that best resembles a particular graph (game model). Obviously, since LBGs are represented as weighted graphs, the approach must consider both connectivity and weights as similarity features.

Tran *et al.* (2016) proposed GpSense, a method that is able to handle massive graphs using data compression and parallelism in GPUs. The approach focuses on optimizing memory issues inherent to previous backtracking methods that allow for a straightforward GPU implementation. It uses weights to encode the maximum degree among nodes and compress large graphs into multiple-level graphs with reduced size.

An analytic approach was developed by Umeyama (1988) to optimize the matching of directed and undirected weighted graphs. The work uses eigendecompositions of adjacency matrices to efficiently find near optimal matches when the graphs are sufficiently close to each other. Regarding time of execution, the method is more tolerant to the combinatorial explosion when compared to exploratory algorithms, however it works only for weighted graphs with the same number of nodes, thus having restrict applications.

Almohamad and Duffuaa (1993) presented a linear programming approach for the WGMP that is solved using a simplex-based algorithm. Although the linear programming formulation presents good performance in matching weighted graphs, its computation time is significantly higher than other approaches.

Bhattacharjee and Jamil (2012) proposed an algorithm called Weighted Subgraph Matching (WSM) to solve the WGMP in large graphs present on biological networks. WSM processes nodes to create a canonical representation that is used in mapping possible matches. These matches are then ranked by cost and only the first k-mappings are evaluated. Consequently,

the remaining matches are discarded to reduce the search space. The algorithm was designed to deliver approximate solutions when handling graphs with up to a thousand nodes, therefore it delivers poor results with smaller graphs.

There are many approaches devoted to tackling GMPs and its variations (CICIRELLO, 1999). However, there are no general algorithms to address all these challenges in polynomial time. Consequently, researchers focus on efficient solutions to particular applications related to these problems. This work addresses the WGMP applied to the balanced transposition of maps of LBGs, and presents three distinct approaches to tackle this issue, an adaptation of the Ullmann's algorithm to work with weighted graphs, a method that uses MCTS to explore the search tree more efficiently, and a GA. Next, works that are related to these approaches are presented.

3.4.1 Ullmann's Algorithm

For being a popular and efficient method to tackle the GMP, the Ullmann's algorithm has been studied and improved over the years. More recently, Blankstein and Goldstein (2010) presented a parallel version of Ullmann's algorithm (implemented in the VFLib library) for execution on multicore machines. The work investigates the advantages and drawbacks of distinct data structures and several heuristics for spawning threads, thus indicating situations where parallelism is clearly superior to single threaded execution.

Lin *et al.* (2012) built on Ullmann's algorithm to improve the refinement by compiling indexes that indicate a more efficient visiting order. In addition, the method includes weights to represent the length of the matching path, therefore extending pruning features.

3.4.2 MCTS

Although MCTS is often used in Artificial Intelligence applications, Maia *et al.* (2017) applied the method to the WGMP. The work is derived from this research and aims at improving game balancing in LBGs. In this thesis, the MCTS algorithm was improved to deliver better results and adapted to perform the transposition of game maps.

In this case, the search space is mapped to a search tree, thus each graph queried from the search space is mapped to a branch of the tree. To optimize the process, the algorithm was adapted to store and rank the best solutions found, so the selection step can use this ranking to guide the exploration of new branches. Chapter 5 details how this process works.

3.4.3 Genetic Algorithms

Many works have used GAs to tackle variants of the GMP. For instance, Auwatanamongkol (2007) used a GA to address inexact graph matching applied to image recognition based on angle matching between two given graphs.

Furthermore, Choi *et al.* (2012) applied a multi-objective genetic algorithm for the subgraph isomorphism problem. The approach included an additional fitness function that considers potentially optimal solutions, thus making the algorithm to operate more efficiently. Xiang *et al.* (2017) also tackled the subgraph isomorphism problem using a dedicated crossover algorithm and a new fitness function that improves the heuristic search.

Some works focus on performance improvement by altering GA attributes and operators. For instance, Khoo and Suganthan (2002) evaluated the results of the GA with distinct crossover operators and different types of individual representation. Singh *et al.* (1997) proposed a new variation of the crossover operator, called the color crossover, and a specific mutation operation to address the problem of structural shape matching.

Finally, Liu *et al.* (1995) addressed the WGMP using a fitness function similar to this work. However, they used a hybrid microgenetic algorithm with a local search algorithm, a modified selection scheme, and a refining procedure to improve the performance of the algorithm in the field of pattern recognition.

3.5 Conclusion

This chapter presented works related to key areas of this thesis. In Section 3.1, the many approaches used to conduct the transposition of LBGs were introduced. However, this work focuses on methods for automatic transposition of maps of LBGs, meaning that minimum or no human intervention is needed.

In Section 3.2, works about game balancing in digital games were shown. Although much effort is devoted to this area, only a few researches focus on LBGs, mainly due to the recent surge of this game genre.

Next, PCG approaches for automatic creation of game levels and game maps were investigated. In the case of LBGs, there is no way to detach game maps from game levels as both constitute the same physical entity. Consequently, challenges and features linked to the usage of PCG with game levels and game maps are related to this work.

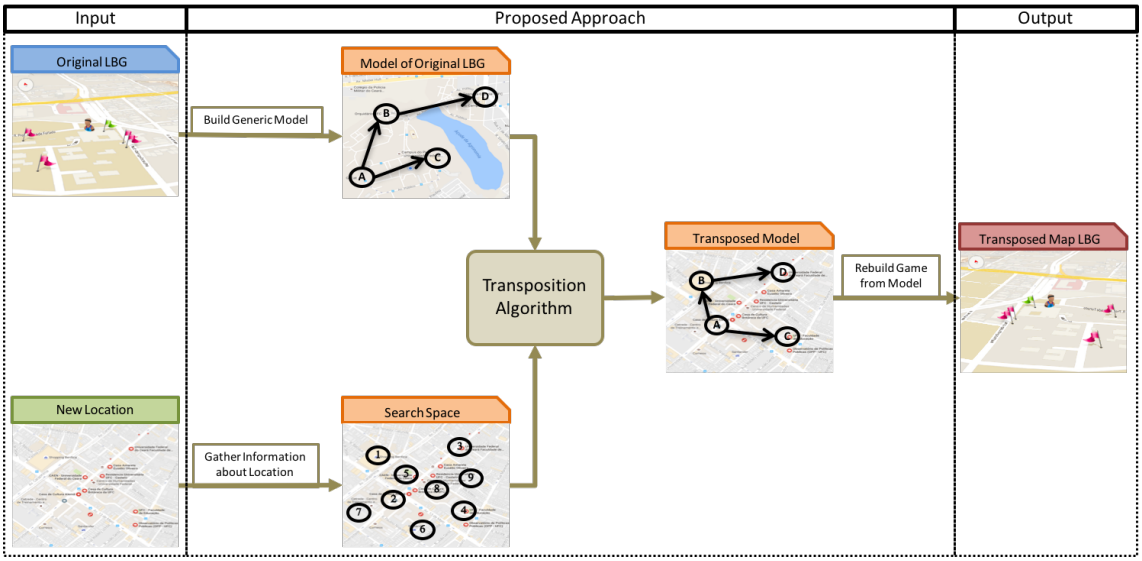
4 BALANCING AND TRANSPOSITION OF MAPS FOR LBGS

This chapter provides an overview of the developed approach to balance and transpose LBGs maps. The goal is to provide a general method capable of handling multiple types of LBGs, estimating game balancing, and automatically generating transposed instances of their maps with similar balancing to the initial game.

In order to achieve this goal, a generic game model was conceived based on a weighted directed graph to represent most LBGs available in the market (Section 4.1). It is also shown how to gather information about the location the LBG will be transposed to, and build the search space according to the proposed game model (Section 4.2). Section 4.3 introduces measurements that focus on estimating game balancing according to particular features of LBGs. These measurements served as the basis upon which the balanced transposition is formulated as an optimization problem applied to the subgraph matching. The problem formulation and the algorithms developed to solve the problem are detailed in Chapter 5. Finally, Section 4.4 concludes the chapter.

In summary, the proposed approach performs according to the following steps: (*i*) build a generic model of the original LBG, (*ii*) gather information about locations in the area where the transposition must take place, (*iii*) apply an algorithm that will minimize game balancing differences in the transposition to the new location, and (*iv*) rebuild the game map from the resulting game model to obtain the transposed instance of the LBG map. Figure 20 presents the previously mentioned pipeline of execution.

Figure 20 – Execution pipeline for the proposed approach.



Source: Author

4.1 Game Model

As mentioned in Chapter 1, this work presents a game model that is capable of representing several types of LBGs using a compact data structure that contains key information for performing a balanced map transposition. The proposed game model aims at generalizing the structure of LBGs regardless of their attributes and game patterns. The main objective of using this game model is to eliminate irrelevant features to the process of maps balancing and transposition, such as graphics, sounds, characters, texts, among others, while maintaining and standardizing all the relevant data required to execute the transposition of maps.

As a result, this model must be flexible, due to the need to represent numerous types of LBGs, but also must convey only the essential data required by the balancing and transposition algorithm. According to these attributes, the proposed game model is based on weighted directed graphs, and can directly encode games that implement the *Search-and-Find* and *Follow-the-Path* patterns. Since these game patterns are present in more than three-quarters of the LBGs, most LBGs on the market can potentially benefit from this work.

Similar representations have been used in (NOLETO *et al.*, 2015) and (MACVEAN *et al.*, 2011) in the development of LBGs. However, both works disregarded the cost to move between places and opted to use unweighted graphs. The balancing and transposition algorithms rely on weights to estimate game balancing and perform transposition, thus being a key feature in this research. This model has successfully been used in (FERREIRA *et al.*, 2019) to demonstrate the transposition of an LBG called “Quest for the Cathedral”.

LBGs are converted into weighted directed graphs using a straight mapping between its components. In this case, the game model can be defined as a graph $G = (V, E, W)$, where V is a set of nodes representing specific locations (places to visit), E is a set of edges ($E \subset V \times V$) symbolizing an existing path between places, and W is the set of edge weights ($W : e_{i,j} \rightarrow \mathbb{N}+$) typifying the cost to move along the path.

Consequently, the adjacency matrix A_G of the weighted graph $G = (V, E, W)$ is a $|V| \times |V|$ matrix defined as:

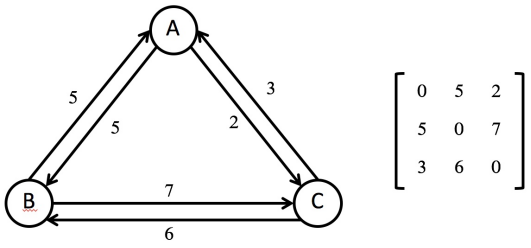
$$A_G = [a_{ij}], where \begin{cases} a_{ij} = W(v_i, v_j), & i \neq j. \\ a_{ij} = 0, & i == j. \end{cases} \tag{4.1}$$

Additionally, adjacency matrices can be extended to include negative weights ($W : e_{i,j} \rightarrow \mathbb{N}$), thus it is possible to set $a_{ij} \leq 0$ if there is no valid path between two distinct locations

i and j. This allows the method to prevent unreachable locations from being selected, as all the weights linked to their edges will be negative. This information is provided by the Abstract Programming Interfaces (APIs) used to build the search space, and will be shown in Section 4.2.

Since G is a weighted directed graph, A_G is a non-symmetric matrix, where rows and columns depict distinct information. For example, Figure 21 illustrates a weighted directed graph and its corresponding matrix. Notice that each row in the matrix contains the weights of edges that are incident out of the corresponding node. Analogously, each column contains the weights of edges that are incident into their corresponding node.

Figure 21 – A weighted directed graph and its adjacency matrix.



Source: Author

Furthermore, nodes can be used to replace geographic coordinates, specific locations, or interaction points in a game, whereas edges are relevant for determining existing paths between nodes, meaning that the absence of an edge $e_{i,j}$ between vertices v_i and v_j indicates there is no virtual path connecting these locations in the game ($W(v_i, v_j) \leq 0$). This is key to encode the flow of the game and its gameplay into the game model. Finally, the weights are fundamental to estimate the game balancing, as well as comparing different maps of the same LBG.

Weights can be associated with distinct characteristics depending on the game. For instance, in competitive games, the relevant aspect to the cost may be the travel time or the distance between two locations; in games that stimulate physical exercise, the cost can be expressed as the calories burned during an activity. Consequently, this information must be provided when creating the game model, as it is crucial to generate a coherent game representation. In Section 4.2, alternatives to estimate the cost accurately are discussed.

In summary, this game model is flexible as edges can be added or excluded to represent custom game flows, and weights can be linked to real world features deemed influential to the game balancing. For example, if an LBG requests players to visit an ordered sequence of locations, the game model will encode a graph with a single chain of edges reaching the

vertices. Furthermore, a fitness game may decide to use footsteps or heartbeats as weights, whereas competitive games can use time, speed, distance, etc. The next section presents the algorithms used to build the corresponding game model for a given LBG and to rebuild the game from a game model.

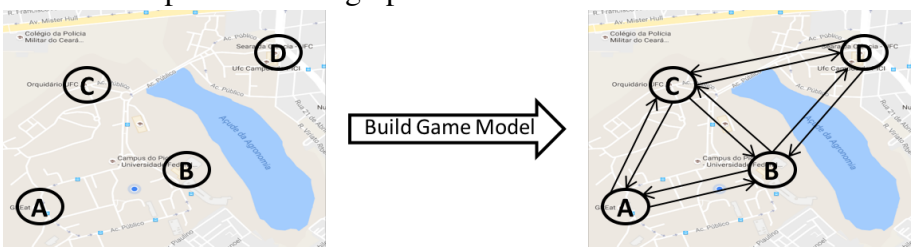
4.1.1 Building the Game Model

Creating a game model (graph $G = (V, E, W)$) for an LBG can be performed by a direct mapping between each visiting location ($l \in L$) in the game ($J = (L, P)$) to a node v ($v \in V$). Then, depending on the game, it is necessary to apply a policy before mapping all the edges. If the game demands players to visit locations according to a fixed order, it is necessary to remove the edges corresponding to the undesired visiting sequences. For example, if the path ($p \in P$) between the nodes v_i and v_j is not valid in the game context, the weight of the respective edge is set to zero ($W(v_i, v_j) = 0$). Usually, LBGs that implement the *Follow-the-Path* game pattern (e.g. *WeQuest*, *FreshUP*, *Tourality*, etc.) present this trait. Conversely, there are some LBGs that do not require players to follow a specific sequence of places to visit (e.g. *Ingress*, *Pokémon GO*, *Parallel Kingdom*, *Floracaching*, etc.), therefore they have their edges and weights mapped directly.

Algorithm 1 builds the game model from an LBG. It has complexity $O(L+P)$ since in the first loop only the locations are processed (L), whereas the second loop creates edges in the graph by traversing all possible paths between locations of the game (P).

Figure 22 showcases a game that does not indicate a specific sequence of locations to visit, therefore players are free to choose any destination, thus the resulting game model will be a complete graph.

Figure 22 – Figure illustrating a case where the game model is a complete directed graph.



Source: Author

Differently, Figure 23 illustrates a simple case where it is necessary to change the

Algorithm 1: Algorithm to build game model from an LBG.

Input: Original LBG $J = (L, P)$

Output: Graph $G = (V, E, W)$

begin

  Create a empty weighted directed graph G;

  for *each location* $l \in L$ **do**

    Create a corresponding node v in V;

  end

  for *each path* p *between two locations* $l_i \in L$ *and* $l_j \in L$ **do**

    Create a corresponding edge $e_{ij} \in E$;

    if $p \notin P$ **then**

      Change its corresponding weight to $w_{ij} = 0$, where $w_{ij} \in W$;

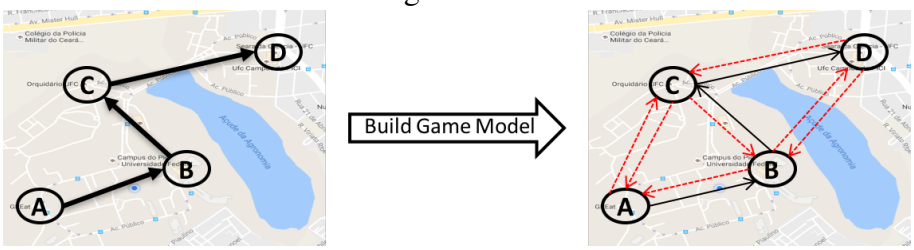
    end

  end

end

game model to correctly represent the flow of the game. In this case, the game stipulates a succession of locations that a player must visit, meaning that some routes were discarded from the gameplay. For instance, players can not go from A to C directly, they must pass before through B. Accordingly, the resulting game model will have the weights of the red edges changed to zero.

Figure 23 – A case where edges have to be removed from the game model to math the game flow.



Source: Author

In summary, the generated game model will always represent the various types of LBGs that use the *Search-and-Find* and the *Follow-the-Path* game patterns using a weighted directed graph. This feature is key to the transposition method, since it ensures that any game model built using this process will have the same format.

4.1.2 Adapting the LBG

As shown in Figure 20, the last step of the transposition process consists in generating an instance of the LBG with the transposed map. This process begins with the transposed game model generated by the transposition algorithm and finishes with the deployment of the game in the new location. Although this process is almost identical to building the game model, it is simpler, as checking for restrictions on paths is no longer necessary.

As a result, creating an instance of the game basically relies on adapting its locations to the corresponding nodes of the game model, maintaining all the previously existing structure. This process can be performed with linear complexity $O(n)$ according to Algorithm 2, which traverses the list of nodes in the resulting game model.

Algorithm 2: Algorithm to adapt the LBG's map according to the transposed game model.

Input: Transposed Graph $G = (V, E, W)$, Original LBG $J = (L_J, P)$
Output: Resulting LBG $R = (L_R, P)$ with the transposed map
begin
    Create a LBG R as a copy of J;
    for *each node* $v \in V$ **do**
        Replace $l_R \in L_R$ by the location of the corresponding node v;
    end
end

The simplicity of these conversions and the low complexity of their algorithms are fundamental to deliver fast transpositions. Chapter 5 will detail some algorithms for the balanced transposition of maps of LBGs, and the next section will address the challenge of estimating game balancing in these games.

4.2 Search Space

As shown in Figure 20, the proposed approach takes a specific region as input to build a search space that will be processed by a transposition algorithm. This search space has to match the game model presented in Section 4.1, as it will provide the new locations to the map of the transposed LBG. Accordingly, the same game model was used to convey this information, thus both game and search space are weighted directed graphs in the form $G = (V, E, W)$.

Building a weighted directed graph for a specific region requires collecting data about places, paths and the cost to move between places. These data are key to define vertices, edges and weights that will constitute a model corresponding to the search space. Therefore, it is necessary to query locations in the area that are of public access and that can be reached by players (e.g. some public places may be closed on weekends, hence they should not be included in the search space when closed).

Nowadays, there are many public APIs that provide a plethora of information about places, also known as POIs (e.g. *Google Places*[1], *Foursquare Venues Service*[2], *Factual Global Places*[3] and *Nominatim*[4]). These APIs offer filtering tools that enable the selection of POIs according to type (e.g. shops, religious places, parks, hospitals, etc.), opening hours, rating (in some cases user ratings are provided), among others. As a result, it can be established a correlation between original and transposed maps of the game. For instance, if the original game requires that players move to a snack bar, in the transposed versions, players can also be required to move to a snack bar in each region where the same game is placed. This correspondence has been successfully implemented in the work developed by Macvean *et al.* (2011), and in the game "Quest for the Cathedral", presented in (FERREIRA *et al.*, 2019).

Once POIs are selected, their geographic coordinates are linked to the vertices of a graph. Another vital and required information is to check for the existence and cost of a path between these locations. As mentioned in Section 4.1, depending on the game, the cost must be linked to distinct features, such as footsteps, heartbeats, time, speed, distance, etc. Consequently, this information is necessary to estimate the costs of the weighted directed graph and build both the game model and the search space. In case the APIs cannot provide information about a specific path or location, their associated weight is defined as negative, so the transposition algorithm can ignore its selection.

Furthermore, the costs experienced by the players during gameplay can be measured using mobile sensors (e.g. footsteps, heartbeats, walking distance, calories, etc.), and the costs used to build the search space can be queried via several APIs that supply estimated information regarding the displacement between two different places (e.g. time, distance, fares, etc.). These APIs usually provide data about existing routes and multiple means of transportation, including walking, car, public transport, bicycle, among others. For instance, the most popular services

[1] https://developers.google.com/places/
[2] https://developer.foursquare.com/overview/venues.html
[3] http://www.factual.com/products/global
[4] http://wiki.openstreetmap.org/wiki/Nominatim

are *Google Distance Matrix*[5], *Mapbox*[6], *GraphHopper*[7], *Microsoft Bing Route Data*[8] and *MapQuest*[9].

The main advantage of using APIs to estimate the costs is the availability of real-time data. Since most services update their database constantly, these APIs provide reliable and precise estimations that also avoid issues such as traffic jams, path routing, road blockages, etc. As a result, using the appropriate features and selecting precise methods to estimate the cost are fundamental to calculate the game balancing properly, since inaccurate costs will consequently lead to incorrect game balancing. For instance, Figure 12 shows that depending on the topography of the cities, using only the distance between POIs to estimate costs can give rise to unbalanced gameplay. Consequently, the data provided by these APIs can be used to estimate key information about the place where the game must be transposed to. Moreover, this dependency on these APIs also poses limitations to the method, as these APIs may not have data about all means of transportation, such as boats, scooters, and others. In this work, four LBGs were designed to serve as a test bed for the proposed balancing and transposition approach, and data queried from *Google Places* and *Google Distance Matrix* was used to estimate game balancing and build game models (Chapter 7).

After querying data about places, paths and costs, it is possible to build the search space as the graph $S = (V, E, W)$. In this case, the selected POIs have their geographic locations linked to vertices V, then the existence of paths between POIs defines the edges E connecting the corresponding vertices, and lastly, the estimated costs for moving along the paths are assigned to the weights W. These steps are presented in Algorithm 3, that has complexity $O(E * E)$.

Regarding the amount of vertices queried from the APIs ($|V_S|$), the only restriction is that the search space must have at least the same amount of vertices ($|V_T|$) of the graph being transposed ($|V_S| \geq |V_T|$), so as to have a bijective mapping $f : V_t \rightarrow V_s$ that represents the isomorphism. Building the search space using this approach usually gives rise to a complete graph, meaning that the adjacency matrix A_S of the search space often has the following form:

$$A_S = [a_{ij}], where \begin{cases} a_{ij} > 0, & i \neq j. \\ a_{ij} \leq 0, & i = j. \end{cases} \tag{4.2}$$

[5] https://developers.google.com/maps/documentation/distance-matrix/
[6] https://www.mapbox.com
[7] https://graphhopper.com
[8] https://msdn.microsoft.com/en-us/library/ff701718.aspx
[9] https://developer.mapquest.com

Algorithm 3: Algorithm to build the model for the search space of a specified region.

Input: Geographic coordinate of the region
Output: Graph $S = (V, E, W)$
begin
    Create an empty weighted directed graph S;
    R_V=Query and select POIs from a location API;
    for *each location* $L \in R_V$ **do**
        Create a corresponding node $v \in V$;
    end
    for *each location* $L_i \in R_V$ **do**
        for *each location* $L_j \in R_V$ *such that* $i \neq j$ **do**
            w_{ij}=Query cost between L_i and L_j;
            if $w_{ij} > 0$ **then**
                Create a corresponding edge $e_{ij} \in E$;
                Change the value of its weight to $w_{ij} \in W$;
            end
        end
    end
end

4.3 Measuring Game Balancing in LBGs

This work focuses on developing PCG methods for the transposition of maps of LBGs while preserving game balancing. Therefore, to assess the effectiveness of the transposition, it is necessary to propose measurements to gauge the game balancing of maps of LBGs. As presented in Section 4.1, LBGs can be converted to a game model based on a weighted directed graph $G = (V, E, W)$, therefore the difficulty level of an LBG in a particular location can be generalized as the total cost to move between the required nodes. Thus, two measurements have been defined to gauge aspects of game balancing in LBGs.

4.3.1 Internal Difficulty Level

The first metric, called Internal Difficulty Level ($\Im$), focuses on assessing the internal balancing of a game, i.e., it evaluates the equality of the costs to move between locations within a game. In this case, unbalanced games have some locations accessible with low cost, whereas

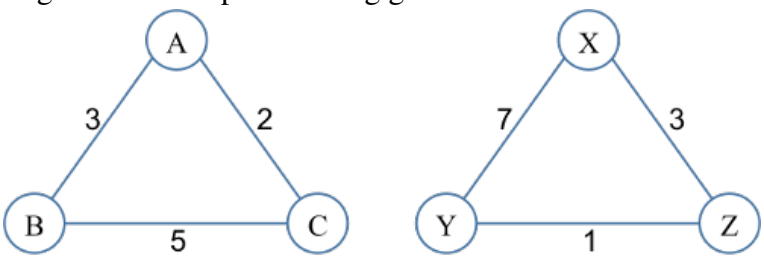
others have a very high cost to be visited. As a result, $\mathfrak{I}$ is defined by the average of the estimated cost linked to every edge in the game model, as shown in Equation 4.3.

$$\mathfrak{I} = \frac{\sum_{x=1}^{|V|}\sum_{y=1}^{|V|} W(V_x,V_y)}{|V|}, \quad W(V_x,V_y) \neq 0. \tag{4.3}$$

where $W(V_x,V_y)$ is the cost to move from node V_x to V_y, and $|V|$ is the number of nodes in the graph. Additionally, the standard deviation ($\sigma_{\mathfrak{I}}$) of $\mathfrak{I}$ is calculated to evaluate the uniformity of node distribution.

From the players' perspective, $\mathfrak{I}$ indicates the average cost to move between locations and $\sigma_{\mathfrak{I}}$ evidences the gap in cost to move along paths. Therefore, higher $\sigma_{\mathfrak{I}}$ means games with greater imbalance between paths. Likewise, if $\sigma_{\mathfrak{I}} = 0$, all paths of a game have the same cost. This measurement is particularly important to investigate whether an LBG oscillates between too easy and too hard challenges, that are deemed undesired according to Schell (2008).

Figure 24 – Graphs showing games with distinct costs.



Source: Author

For instance, Figure 24 shows graphs that are slightly different in cost, but the graph to the left has $\mathfrak{I}_L = 3.333$ and $\sigma_{\mathfrak{I}_L} = 1.247$, whereas the graph to the right has $\mathfrak{I}_R = 3.666$ and $\sigma_{\mathfrak{I}_R} = 2.494$. This indicates the latter has a bigger disparity in its costs, as corroborated by the weights $W(v_y,v_x) = 7$ and $W(v_y,v_z) = 1$.

4.3.2 Minimum Balancing Difference

The second metric is called Minimum Balancing Difference ($\mathfrak{M}$), and was developed to highlight dissimilarities in game balancing between instances of the same game played in different areas. This measurement calculates the minimum difference in the game balancing considering each path that composes the games. In this case, since LBGs are mapped to weighted graphs, the analysis is equivalent to the graph matching presented in (BHATTACHARJEE; JAMIL, 2012). This process extracts the best similarity between the paths of both games by minimizing their differences. $\mathfrak{M}$ is given by the sum of the differences between all the

corresponding paths in the best similarity case, as expressed by Equation 4.4. Next, the details on how to calculate $\mathfrak{M}$ are shown.

Consider an LBG that can be played in two distinct areas, thus giving rise to two different game configurations (G_A and G_B). $\mathfrak{M}$ is obtained by matching a set of nodes A of G_A to a set of nodes B of G_B so that:

$$\mathfrak{M} = min \sum_{x=1}^{|V_A|} \sum_{y=1}^{|V_B|} |W_A(V_{Ax}, V_{Ay}) - W_B(V_{Bx}, V_{By})| \tag{4.4}$$

where $W_A(V_{Ax}, V_{Ay})$ and $W_B(V_{Bx}, V_{By})$ represent the weights of paths connecting nodes V_{Ax} to V_{Ay} and V_{Bx} to V_{By}, respectively.

For the sake of simplicity, the following example demonstrates how to calculate $\mathfrak{M}$ for two weighted graphs containing only three vertices. Consider that Figure 24 depicts graphs representing the same LBG when played in two areas with distinct difficulty levels. Despite having only three vertices, there are many possible matches between these graphs. For instance, the mapping $(A/X, B/Y, C/Z)$ presents the following difference between paths $|3-7|+|2-3|+|5-1| = 9$, however the best match is $(A/Z, B/X, C/Y)$, thus the Minimum Balancing Difference is 3 ($\mathfrak{M} = |3-3|+|2-1|+|5-7|$).

In resume, $\mathfrak{M}$ indicates whether players in different locations can compete more fairly since it shows how unbalanced the games are, considering each available path. Therefore, if $\mathfrak{M} = 0$ for two distinct maps, players in both areas should experience an equivalent game balancing because the estimated cost to move between each location is equal in both games. Next chapter presents the formulation of the transposition challenge as an optimization problem that aims at minimizing $\mathfrak{M}$.

4.4 Conclusion

This chapter detailed the phases that comprise the approach to balance and transpose LBGs. First, a broad overview of the method illustrating its execution pipeline was provided, then the game model used to represent multiple types of LBGs was presented (Section 4.1). Next, Section 4.2 detailed the means to build the search space according to the proposed game model, including the use of popular APIs available freely on the Internet. Lastly, Section 4.3 discussed the measurements devised to assess game balancing internally (Section 4.3.1) and to compare game balancing between two instances of a game (Section 4.3.2).

A key part in this process is the transposition algorithm, since it is the step in charge of selecting the locations that will constitute the transposed game map while focusing on maintaining the game balancing. The next chapter details the formulation of this challenge and three algorithms developed to accomplish this task.

5 PROBLEM FORMULATION AND ALGORITHMS

In this chapter the challenge of transposing maps of LBGs as a WGMP is presented, and Section 5.1 builds on $\mathfrak{M}$ to elaborate algorithms to address this challenge as an optimization problem. As mentioned in Chapter 2, the subgraph isomorphism problem is a complex challenge that grows exponentially with the size of the graphs. Therefore three distinct algorithms were extended to tackle this issue for LBG transposing. Each algorithm has particular attributes that are suited to the many configurations a game may have. The following methods are presented: an algorithm based on the MCTS (Section 5.2), a deterministic approach that extends the Ullmann's algorithm to work on weighted graphs (Section 5.3), and a Genetic Algorithm (Section 5.4). Lastly, Section 5.5 summarizes and concludes this chapter.

5.1 Problem Formulation

As depicted in Figure 20, the transposition algorithm takes two game models as input, one representing the original version of the game - here called the target model $G_T = (V_T, E_T, W_T)$ - and another characterizing the search space $G_S = (V_S, E_S, W_S)$. The last is built using information gathered from the area where the map is being transposed to (as shown in Section 4.2). The purpose of the transposition algorithm is to create a bijective mapping between locations from the original game and the search space ($\mathscr{F} : V_T \rightarrow V_S$), thus addressing the challenge as a WGMP. Consequently, the resulting game model $G_R = (V_R, E_R, W_R)$ is composed of transposed locations selected from the search space ($V_R \subset V_S, E_R \subset E_S$, and $W_R \subset W_S$).

The authors build on the concept of root matrix M presented in Section 2.8.1 to manage the distinct solutions. In this case, M is a $|V_T| \times |V_S|$ integer matrix that encodes mappings $V_T \rightarrow V_S$ according to the following rule:

$$m_{ij} = \begin{cases} 1, \text{ if } v_i \in T \text{ is mapped to } v_j \in S \\ 0, \text{ otherwise} \end{cases}$$

Consequently, M must originate a set of integer matrices $P = \{P_1, P_2, ..., P_n\}$ (each with size $|V_T| \times |V_S|$) containing the many selections of vertices and permutations that create bijective mappings corresponding to a single solution. In this case, to secure an exact one-to-one mapping between V_R and V_S, each matrix $P_n \in P$ must have their elements (p_{ij}) defined as:

$$P_n = [p_{ij}], where \begin{cases} \sum_{j=0}^{|V_S|} p_{ij} = 1, & \sum_{i=0}^{|V_T|} . \\ \sum_{i=0}^{|V_R|} p_{ij} \leq 1, & \sum_{j=0}^{|V_S|} . \end{cases} \tag{5.1}$$

In resume, the rules presented above make sure that each line contains exactly one "1" and that each column contains at most one "1", hence the transposition algorithm must narrow the search space down by avoiding unsuitable solutions (as shown in Figure 15) while focusing on more promising candidates from P.

Ideally, the transposition algorithm will produce a transposed map where the cost to move between each location has the same cost of the corresponding path in the original map. Thus the ideal transposed map would satisfy the following relation:

$$\sum_{v_x \in V_T} \sum_{v_y \in V_T} (W_T(v_x, v_y) - W_R(\mathscr{F}(v_x), \mathscr{F}(v_y)) = 0 \tag{5.2}$$

However, the cost to move between locations in distinct regions is seldom equal, therefore the transposition algorithm must operate to ensure the game balancing between G_T and G_R to be as similar as possible. Section 4.3.2 presented how $\mathfrak{M}$ calculates the minimum difference in the game balancing between distinct game instances. $\mathfrak{M}$ looks for the permutation of locations that better matches the game balancing between two game models, hence it was used as the chief attribute to compel the transposition algorithms to minimize differences in game balancing between G_T and G_R. In this case, the mapping between locations from the original game and the search space must consider the selection of vertices from G_S that will constitute G_R, and the permutation of vertices that yield minimum differences in game balancing. As a result, this challenge is formulated as an optimization problem by defining a cost function $\mathfrak{T}$ that must be minimized to make G_R as similar as possible to G_T.

To elaborate this problem, consider the weighted directed graphs G_T and G_S, and their respective adjacency matrices A_T (size $|V_T| \times |V_T|$) and A_S (size $|V_S| \times |V_S|$), defined according to the Equation 4.1 presented in Section 4.1. The transposition algorithm works as a bijective mapping $\mathfrak{T} : V_T \rightarrow V_S$ such that the resulting graph G_R is built from the minimization of $\mathfrak{T}(A_T, A_S)$. Since P_n (Equation 5.1) maps a particular solution for the isomorphism problem, the adjacency matrix A_R of G_R can be defined as:

$$A_R = P_n * A_S * P_n^T. \tag{5.3}$$

On top of that, to calculate the difference in game balancing A_R must be conformed to generate A'_R, an adjacency matrix containing solely the weights of existing paths in A_T. The proper selection of paths in A_R can be performed by generating a filtering matrix (F) using Boolean operations on A_T (Equation 5.4), and further applying F to A_R with the Hadamard

product (also known as the entrywise product or the Schur product).

$$F = J \wedge A_T, \text{ where } J \text{ is an all-ones matrix with the same size of } A_T. \tag{5.4}$$

$$A'_R = F \cdot A_R, \text{ where } \cdot \text{ is the Hadamard product.} \tag{5.5}$$

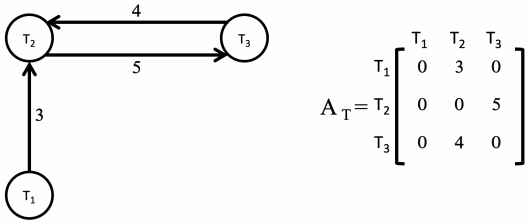
Finally, the cost function $\mathfrak{T}$ can be formulated as:

$$\mathfrak{T}(A_T, A_S) = \mathfrak{S}(|\, A_T - A'_R \,|), \text{ where } \mathfrak{S} \text{ calculates the grand sum of a matrix.} \tag{5.6}$$

5.1.1 Example

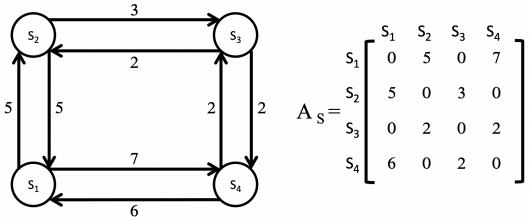
In this section, a simplified example of the game and the search space is presented to illustrate how the transposition algorithm makes use of the formulation previously shown. Given two game models, one portraying the original instance of the game G_T (Figure 25) and another representing the search space G_S (Figure 26), along with their respective adjacency matrices A_T and A_G.

Figure 25 – An example of target game model and its adjacency matrix.



Source: Author

Figure 26 – A game model depicting an example of search space and its adjacency matrix.



Source: Author

The transposition algorithm must build a game model G_R from a set of possible solutions to the isomorphism problem. Thus, the algorithm could inspect candidate solutions

thoroughly. As an example, consider only two candidate matrices P_1 (that maps T_1/S_1, T_2/S_2, and T_3/S_3) and P_2 (that maps T_1/S_3, T_2/S_2, and T_3/S_1), defined as follows:

$$P_1 = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{vmatrix}, P_2 = \begin{vmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{vmatrix}$$

In this case, the algorithm must select among P_1 and P_2 the solution that minimizes the differences in game balancing. Hence, it is necessary to submit each solution to the Equation 5.6, then the resulting game model G_R is generated from a mapping that presents the lower game balancing difference.

For the sake of simplicity, first it is shown the calculations of the adjacency matrices A_{R_1} and A_{R_2} that represent P_1 and P_2, respectively.

$$A_{R_1} = P_1 * A_S * P_1^T$$

$$A_{R_1} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{vmatrix} * \begin{vmatrix} 0 & 5 & 0 & 7 \\ 5 & 0 & 3 & 0 \\ 0 & 2 & 0 & 2 \\ 6 & 0 & 2 & 0 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{vmatrix}$$

$$A_{R_1} = \begin{vmatrix} 0 & 5 & 0 \\ 5 & 0 & 3 \\ 0 & 2 & 0 \end{vmatrix}$$

$$A_{R_2} = P_2 * A_S * P_2^T$$

$$A_{R_2} = \begin{vmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{vmatrix} * \begin{vmatrix} 0 & 5 & 0 & 7 \\ 5 & 0 & 3 & 0 \\ 0 & 2 & 0 & 2 \\ 6 & 0 & 2 & 0 \end{vmatrix} * \begin{vmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{vmatrix}$$

$$A_{R_2} = \begin{vmatrix} 0 & 2 & 0 \\ 3 & 0 & 5 \\ 0 & 5 & 0 \end{vmatrix}$$

Next, each step of the process that calculates $\mathfrak{T}(A_T, A_S)$ considering the mapping defined in P_1 is shown:

$$\mathfrak{T}(A_T, A_S) = \mathfrak{S}(|\, A_T - A'_{R_1} \,|)$$

$$\mathfrak{T}(A_T, A_S) = \mathfrak{S}(|\, A_T - F \cdot A_{R_1} \,|)$$

$$\mathfrak{T}(A_T, A_S) = \mathfrak{S}(|\, A_T - F \cdot \begin{vmatrix} 0 & 5 & 0 \\ 5 & 0 & 3 \\ 0 & 2 & 0 \end{vmatrix} |)$$

$$\mathfrak{T}(A_T, A_S) = \mathfrak{S}(|\, A_T - (J \wedge A_T) \cdot \begin{vmatrix} 0 & 5 & 0 \\ 5 & 0 & 3 \\ 0 & 2 & 0 \end{vmatrix} |)$$

$$\mathfrak{T}(A_T, A_S) = \mathfrak{S}(|\, A_T - (\begin{vmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix} \wedge \begin{vmatrix} 0 & 3 & 0 \\ 0 & 0 & 5 \\ 0 & 4 & 0 \end{vmatrix}) \cdot \begin{vmatrix} 0 & 5 & 0 \\ 5 & 0 & 3 \\ 0 & 2 & 0 \end{vmatrix} |)$$

$$\mathfrak{T}(A_T, A_S) = \mathfrak{S}(|\, A_T - \begin{vmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{vmatrix} \cdot \begin{vmatrix} 0 & 5 & 0 \\ 5 & 0 & 3 \\ 0 & 2 & 0 \end{vmatrix} |)$$

$$\mathfrak{T}(A_T, A_S) = \mathfrak{S}(|\, \begin{vmatrix} 0 & 3 & 0 \\ 0 & 0 & 5 \\ 0 & 4 & 0 \end{vmatrix} - \begin{vmatrix} 0 & 5 & 0 \\ 0 & 0 & 3 \\ 0 & 2 & 0 \end{vmatrix} |)$$

$$\mathfrak{T}(A_T, A_S) = \mathfrak{S}(|\, \begin{vmatrix} 0 & -2 & 0 \\ 0 & 0 & 2 \\ 0 & 2 & 0 \end{vmatrix} |)$$

$$\mathfrak{T}(A_T, A_S) = \mathfrak{S}(\begin{vmatrix} 0 & 2 & 0 \\ 0 & 0 & 2 \\ 0 & 2 & 0 \end{vmatrix})$$

$$\mathfrak{T}(A_T, A_S) = 6.$$

Similarly, the same calculations must be performed to obtain the game balancing difference for the mapping defined in P_2:

$$\mathfrak{T}(A_T, A_S) = \mathfrak{S}(\mid A_T - A'_{R_2} \mid)$$

$$\mathfrak{T}(A_T, A_S) = \mathfrak{S}(\mid A_T - F \cdot A_{R_2} \mid)$$

$$\mathfrak{T}(A_T, A_S) = \mathfrak{S}(\mid A_T - F \cdot \begin{vmatrix} 0 & 2 & 0 \\ 3 & 0 & 5 \\ 0 & 5 & 0 \end{vmatrix} \mid)$$

$$\mathfrak{T}(A_T, A_S) = \mathfrak{S}(\mid A_T - (J \wedge A_T) \cdot \begin{vmatrix} 0 & 2 & 0 \\ 3 & 0 & 5 \\ 0 & 5 & 0 \end{vmatrix} \mid)$$

$$\mathfrak{T}(A_T, A_S) = \mathfrak{S}(\mid A_T - (\begin{vmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix} \wedge \begin{vmatrix} 0 & 3 & 0 \\ 0 & 0 & 5 \\ 0 & 4 & 0 \end{vmatrix}) \cdot \begin{vmatrix} 0 & 2 & 0 \\ 3 & 0 & 5 \\ 0 & 5 & 0 \end{vmatrix} \mid)$$

$$\mathfrak{T}(A_T, A_S) = \mathfrak{S}(\mid A_T - \begin{vmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{vmatrix} \cdot \begin{vmatrix} 0 & 2 & 0 \\ 3 & 0 & 5 \\ 0 & 5 & 0 \end{vmatrix} \mid)$$

$$\mathfrak{T}(A_T, A_S) = \mathfrak{S}(\mid \begin{vmatrix} 0 & 3 & 0 \\ 0 & 0 & 5 \\ 0 & 4 & 0 \end{vmatrix} - \begin{vmatrix} 0 & 2 & 0 \\ 0 & 0 & 5 \\ 0 & 5 & 0 \end{vmatrix} \mid)$$

$$\mathfrak{T}(A_T, A_S) = \mathfrak{S}(\mid \begin{vmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{vmatrix} \mid)$$

$$\mathfrak{T}(A_T, A_S) = \mathfrak{S}(\begin{vmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{vmatrix})$$

$$\mathfrak{T}(A_T, A_S) = 2.$$

In this example, P_2 originates the mapping (T_1/S_3, T_2/S_2, T_3/S_1) and presents lower $\mathfrak{T}$ than P_1, meaning that a transposed map containing the mapping of P_2 will yield minor differences in game balancing. After defining the selection that originates the minimum $\mathfrak{T}$, the last step in the transposition is to build the transposed game model by replacing the locations of the original game by the ones of the resulting solution.

5.1.2 Discussion

The main adversity to the formulation previously shown is that both the original game map and the search space can originate game models with varying size. Besides, the larger a search space is, the more likely it will contain locations that resemble the game balancing present in the original game. Thus, the transposition algorithm must cope with distinct configurations that can require huge processing capabilities.

In theory, the transposition algorithm should select the best solution from the many candidates, however, in practice, the time and effort to process all solutions increase exponentially with the size of G_T and G_S according to the formula of k-permutations without repetitions (where $n = |V_S|$ and $k = |V_T|$):

$$P_{n,k} = \frac{n!}{(n-k)!}. \tag{5.7}$$

As a result, the amount of possible solutions quickly reaches a point that makes brute-force approaches inappropriate. Consequently, the next sections present three distinct algorithms conceived to handle the broad range of game models according to their sizes.

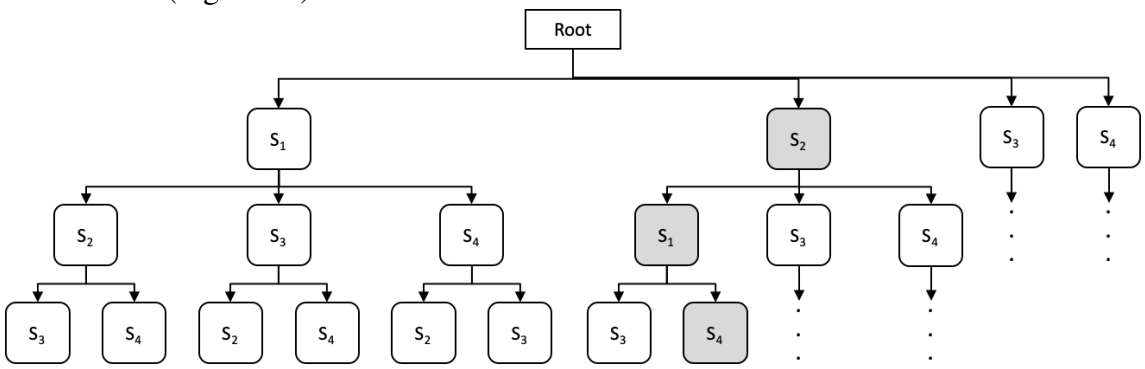
5.2 Monte Carlo Tree Search

In this work, an algorithm based on the well known MCTS is presented. MCTS is widely applied to explore large search trees due to its ability to use data from previously processed solutions to guide the search for better ones. Besides, a key advantage of MCTS is that the method operates under a predefined computational budget (usually, time, memory or number of iterations), so the algorithm outputs the best solution found within certain constraints. MCTS consists of four steps, namely *Selection*, *Expansion*, *Simulation*, and *Backpropagation*. Consequently, this section details the adaptations made in each step of the approach. An early implementation of this method was first introduced in (MAIA *et al.*, 2017) and (FERREIRA *et al.*, 2019), however, adjustments have been made to improve the quality of its results.

First, bear in mind that MCTS is regarded as an artificial intelligence algorithm since it improves the knowledge about the problem with each interaction. Therefore, in the beginning of execution, the method has no hint on how to explore the search tree, then it starts by choosing branches randomly. As further interactions are processed, the algorithm "learns" that some sections of the tree generate unfitting solutions, while others create better ones, thus the approach focuses on exploring branches that are more promising (this is the *Selection* step).

The learning procedure relies on a partial tree to store the data about the branches and solutions explored. Each node in the partial tree represents a mapping between vertices of the original game and the search space. For instance, Figure 27 shows a search tree built to illustrate the mapping between graphs G_T (Figure 25) and G_S (Figure 26), and the highlighted branch encodes the mapping (T_1/S_2, T_2/S_1, T_3/S_4). These nodes are vital to the algorithm since they hold a record encoding the amount of good and bad solutions found below them, in the partial tree.

Figure 27 – Search tree illustrating the mapping between graphs G_T (Figure 25) and G_S (Figure 26).



Source: Author

Consequently, a leaf on the partial tree depicts the last mapping between vertices of the original game and search space, so the complete path from the root to a leaf spawns a solution to the problem. The process of building solutions out of unexplored paths is called *Expansion*. Since processing all leaves is equivalent to cracking the problem using brute-force, the purpose of using MCTS is to use the information encoded in the nodes of the partial tree to look for branches that generate the best solutions.

Once a complete solution is created, the algorithm must decide whether it is a good or bad result. This process is called *Simulation* as it intends to reproduce the outcome of choosing this solution as the best one. In this work, *Simulation* consists in calculating $\mathfrak{M}$ and comparing it to the best solutions found earlier. In this case, a list containing a percentage (κ) of all branches

evaluated is constantly updated to store the best solutions found. Therefore, if a solution enters the list it is deemed good, otherwise it is considered bad.

Finally, in order to update the records of nodes in the partial tree, the step called *Backpropagation* is started. Consequently, for every solution explored, its nodes will be updated to indicate the quality of the solutions found below them.

However, even though a specific section of the tree may constantly generate good results, the best solution can be in an unexplored area of the tree, therefore the method may eventually diversify explored branches to avoid local minimum. As a result, the algorithm must decide whether to focus exclusively on promising branches (exploitation) or to investigate new ones (exploration). The reasoning behind choosing which branch to investigate is referred as *tree policy*. The *exploitation-exploration dilemma* is the decision between continuing to exploit branches that are believed to be optimal, or starting to explore other branches that are currently sub-optimal but may occasionally contain better results. This problem is also linked to Bandit problems (JUN, 2004), and are addressed by decision policies that intend to minimize regret. This work uses Upper Confidence Bounds for Trees (UCT) as the tree policy due to its simplicity and efficiency to solve the exploration-exploitation dilemma (KOCSIS; SZEPESVáRI, 2006).

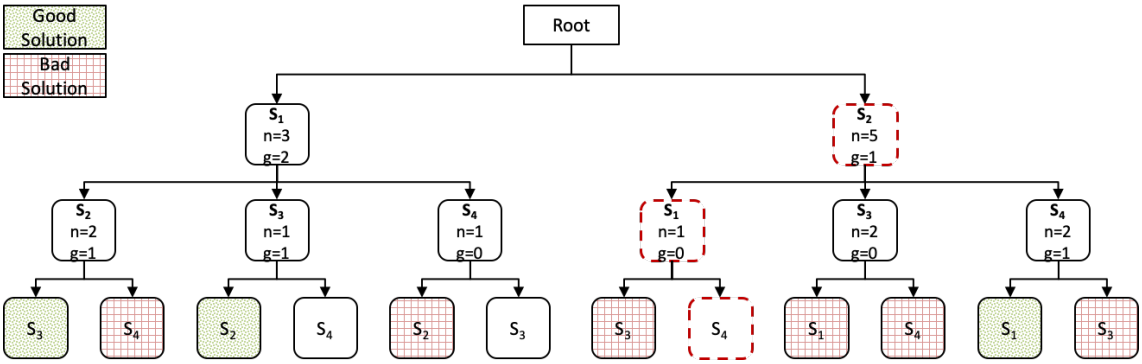
UCT selects nodes based on their probability to be part of an optimal solution. This prospect for a node x is given by Equation 5.8, that uses the average of rewards $\bar{A}_x$, the number of visits in the parent node (n_p), the number of times the node x has been visited (n_x), and a constant C (where $C > 0$). In summary, Equation 5.8 shows that UCT equates exploitation and exploration, since the denominator of the exploration term (n_x) increases with each visiting, hence lowering its contribution. Conversely, if a node is visited, the numerator of sibling nodes (n_p) rises, which increases exploitation. Besides, this property ensures that, given enough iterations, even sub-optimal nodes may be selected.

$$UCT(x) = \bar{A}_x + 2C\sqrt{\frac{2\ln(n_p)}{n_x}} \tag{5.8}$$

During *Backpropagation*, MCTS updates the values of UCT for each node previously simulated. As a result, the algorithm must recalculate the new values of UCT when performing *Selection*. This feature encodes the "learning" process by recalling the outcome of previous solutions. To showcase this process, consider the partial tree depicted in Figure 28. Notice that each node x holds the data necessary to calculate UCT. In this case, the amount of times it has

been visited (n_x), and the number of good solutions below them in the partial tree (g_x) are used to calculate $\bar{A}_x = \frac{g_x}{n_x}$. For example, once the *Simulation* decides whether the highlighted mapping (T_1/S_2, T_2/S_1, T_3/S_4) is a good or bad solution, the data stored in nodes S_2 and S_1 are updated accordingly.

Figure 28 – Depiction of a partial tree and the data stored in each node.



Source: Author

This process is repeated until a certain budget is reached, and the best solution processed during execution is selected. A final parallel step was added to check for better results in the automorphism of this solution, as shown in Algorithm 4.

However, calculating the best solution in the automorphism for a graph is exponential to its size, thus the algorithm is not suited for large graphs. As a result, the complexity of the MCTS algorithm depends on the budget and the number of vertices in the graphs G_T and G_S as processed by each step. Accordingly, the selection and backpropagation steps have complexity $O(B|V_S|)$, where B is the branching factor of the tree, the simulation and expansion have complexity $O(|V_T|)$, the management and sorting of the list L has complexity $O(|L|log_{|L|})$, and finally, the parallel method to check for the automorphism has complexity $O(|V_T|!)$. Thus, the overall complexity of the algorithm is $O(\mathscr{B}(|V_T| + |V_S|B + |L|log_{|L|} + |V_T|!))$.

Besides, given that randomness is inherent to MCTS, it is classified as a non-deterministic approach, so results can vary between executions even if the same input is used. Next, an example to illustrate how each step works is presented.

5.2.1 Example

To showcase how MCTS was adapted to solve the graph isomorphism problem, consider the game model shown in Figure 25 to be G_T, and the game model depicted in Figure 26 to be G_S. For the sake of simplicity, let's assume that the list L can hold only one solution,

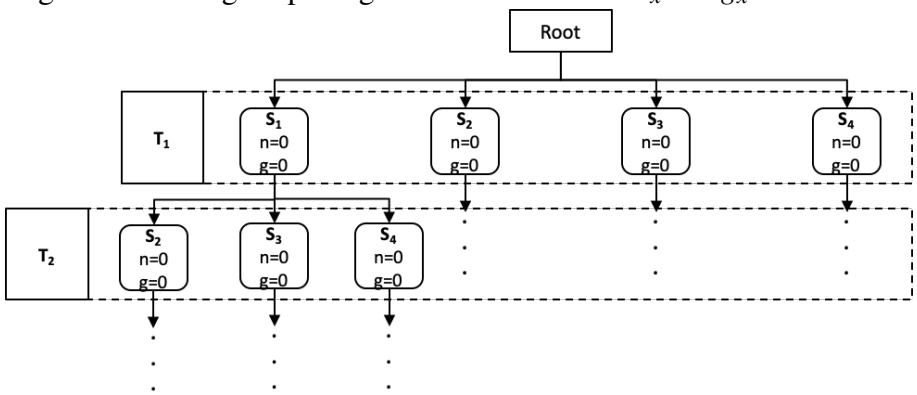
Algorithm 4: Algorithm describing MCTS steps.

Input: Matrix A_T, Matrix A_S, Best percentage κ, Budget $\mathscr{B}$
Output: A matrix P_R that encodes the best solution found
begin
  Create the root matrix M to encode the search tree;
  Create matrices N and G with the dimensions of M;
  Initialize the elements of N and G with value 0;
  Create a list L with size corresponding to $\kappa * \mathscr{B}$;
  Initialize the elements of L with value ∞;
  while *Solutions processed* < B **do**
    Calculate UCT and select nodes with higher values;
    Expand branches that have not been processed;
    Simulate the selected solution p by calculating $\mathfrak{M}_p$;
    if $\mathfrak{M}_p >$ *worst solution in* L **then**
      Classify solution as bad;
    else
      Classify solution as good;
      Remove the last element of L;
      Insert $\mathfrak{M}_p$ in L;
    end
    Backpropagate the status of p in matrices N and G;
  end
  Select the best solution in L and check for its automorphism;
  Build matrix P_R from to the best solution found;
end

therefore the evaluated solutions are considered good only if they are better than the ones investigated earlier.

In the beginning, the values of n and g for each node x are set to 0 (Figure 29), and the algorithm has not performed any exploration to the search tree. Hence the selection of the first solution is made randomly. Supposing this first solution resulted in the mapping (T_1/S_1, T_2/S_2, T_3/S_3), the algorithm calculates $\mathfrak{M} = 6$ for this mapping, and since the initial value in L is ∞, this result is regarded as a good solution. This step is equivalent to *Simulation*, hence its outcome must be spread throughout the partial tree (*Backpropagation*) to guide further selections. At the end of this first round, the values of n and g for each node are shown in Figure 30.

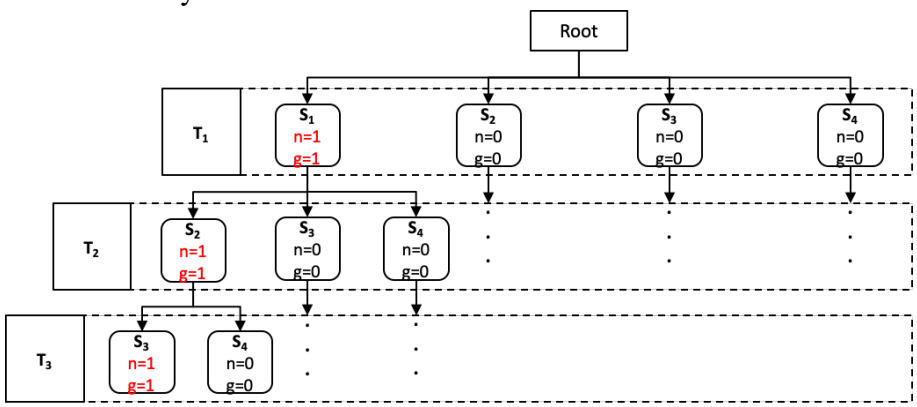
Figure 29 – Image depicting the initial values for n_x and g_x in each node.



Source: Author

Similarly, if the next solution to investigate is the mapping (T_1/S_1, T_2/S_2, T_3/S_4), the algorithm fails to calculate $\mathfrak{M}$ because there are no edges connecting S_2 to S_4 in the game model (Figure 26). Evidently, the mapping is regarded as not good, so the values of n are updated accordingly, as shown in Figure 31. Furthermore, if a mapping between two particular nodes of G_T and G_S is invalid or always generates bad outcomes, all solutions below the node containing this mapping will naturally be unsuitable, so given enough processing rounds, the algorithm will recognize this feature and begin avoiding to explore this branch due to their low UCT.

Figure 30 – Values for n and g are updated at the end of each MCTS cycle.

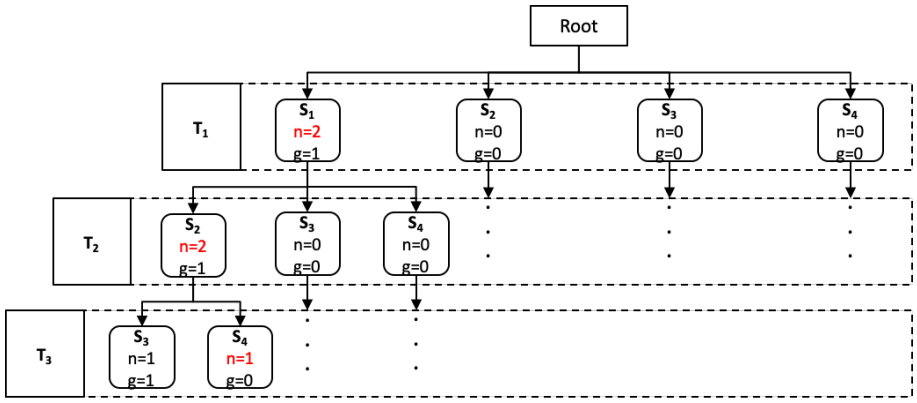


Source: Author

Now, consider that MCTS has processed many solutions, up to the point that the algorithm starts to "learn" about the search tree. In this case, the *Selection* step applies the tree policy to guide further explorations. For example, given that Figure 32 depicts the values stored in the nodes of a partial tree, the algorithm must calculate $UCT(x)$ for each node x. In this case, Table 3 shows these operations, considering $C = 1$:

Notice that, albeit S_1 has managed to find good solutions in most cases ($g_{S_1} = 4$ out

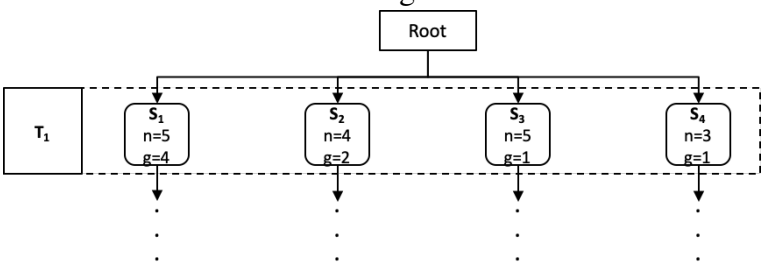
Figure 31 – Values for n are updated to indicate a sub-optimal solution has been found.



Source: Author

of $n_{S_1} = 5$), $UCT(S_4)$ has the highest value because it is currently the least explored branch. This property allows the algorithm to avoid minimum locals by balancing the relation exploration-exploitation, as the computation of UCT for a node is determined by a combination of past results (average reward $\bar{A}$) and the amount of solutions explored with respect to sibling nodes ($\ln(n_p)$). Besides, this balancing between exploration and exploitation can be adjusted by the constant C. In general, if C is closer to 0, it reduces the contribution of exploration to the value of UCT, whereas if C has higher values, the influence of exploration rises.

Figure 32 – Figure presenting an example of node values for selection during execution of MCTS.



Source: Author

Table 3 – Table detailing the operations to calculate $UCT(x)$ for each node shown in Figure 32 for $C = 1$.

$UCT(S_1)$	$UCT(S_2)$	$UCT(S_3)$	$UCT(S_4)$
$= \bar{A}_{S_1} + 2C\sqrt{\frac{2\ln(n_p)}{n_{S_1}}}$	$= \bar{A}_{S_2} + 2C\sqrt{\frac{2\ln(n_p)}{n_{S_2}}}$	$= \bar{A}_{S_3} + 2C\sqrt{\frac{2\ln(n_p)}{n_{S_3}}}$	$= \bar{A}_{S_4} + 2C\sqrt{\frac{2\ln(n_p)}{n_{S_4}}}$
$= \frac{g_{S_1}}{n_{S_1}} + 2\sqrt{\frac{2\ln 17}{5}}$	$= \frac{g_{S_2}}{n_{S_2}} + 2\sqrt{\frac{2\ln 17}{4}}$	$= \frac{g_{S_3}}{n_{S_3}} + 2\sqrt{\frac{2\ln 17}{5}}$	$= \frac{g_{S_4}}{n_{S_4}} + 2\sqrt{\frac{2\ln 17}{3}}$
$= \frac{4}{5} + 2\sqrt{\frac{5.66}{5}}$	$= \frac{2}{4} + 2\sqrt{\frac{5.66}{4}}$	$= \frac{1}{5} + 2\sqrt{\frac{5.66}{5}}$	$= \frac{1}{3} + 2\sqrt{\frac{5.66}{3}}$
$= 2.93$	$= 2.88$	2.33	3.08

Source: Author

After selecting a solution within the branch of S_3, let's assume that an invalid or bad mapping was found. Accordingly, the values of n and g have to be updated and the new value of UCT must be calculated ($UCT(S_3) = 2.63$). Then, in the next cycle, the algorithm will choose a solution containing S_1, since $UCT(S_1) = 2.93$ will be the highest value.

This process is repeated until the budget is reached. The last step of the method consists in selecting the best result found among all the processed solutions, and checking for its automorphism. For instance, lets suppose the best solution processed by MCTS is the mapping (T_1/S_1, T_2/S_2, T_3/S_3) which has $\mathfrak{M} = 6$. The method then checks for all of its permutations, including the mapping (T_1/S_3, T_2/S_2, T_3/S_1), that delivers the best automorphic solution ($\mathfrak{M} = 2$), thus being returned as the ultimate solution.

5.3 Parallel Weighted Ullmann

Section 3.4.1 presented the algorithm proposed by Ullmann to tackle the GMP. The approach combines a pruning analysis with a depth first tree search to eliminate inadequate solutions from the search space. The key feature in the refinement proposed by Ullmann is the connectivity information linked to each vertex of the graph, called degree. In summary, the method compares the degree (number of edges $e_T \in E_T$) of a vertex $v_T \in V_T$ with the degree (number of edges $e_S \in E_S$) of a vertex $v_S \in V_S$ to exclude all possible solutions that map $v_T \rightarrow v_S$ if the degree of v_T is bigger than the degree of v_S. As a result, the effectiveness of the refinement relies on the difference of connectivity between vertices, therefore no refinement occurs and the algorithm explores all possible solutions when submitted to complete graphs.

Despite being widely referred and used in many applications, Ullmann's algorithm is not suited to tackle the challenge addressed in this work due to two key characteristics. First, Ullmann's algorithm was designed to work with undirected unweighted graphs, and second, the search space built using the information gathered from Internet APIs is usually a complete graph, as mentioned in Section 4.2. In this case, an adaptation of the original Ullmann's algorithm called *Parallel Weighted Ullmann* (PWU) was conceived. It processes weighted graphs to implement the refinement process.

The proposed algorithm alters the refinement of the search space to make the degree of each vertex include a restriction based on the difference between the weights of v_T and v_S. Hence, to define the degree of a vertex v_S it is necessary to sum the number of its edges e_S and deduct eventual edges that have weight w_S too distinct from the weights w_T of v_T. This

approach can be implemented by linking a threshold $\tau \in \mathbb{R}^+$ to each edge to define an acceptable percentage of similarity defined by the relation:

$$| w_T - w_S | \leq \tau * w_T \tag{5.9}$$

Accordingly, it is possible to adjust the value of τ to prune the search space. For example, if $\tau > 0$, the degree of a vertex v_S with respect to a vertex v_T is calculated by the number of edges e_S that have their weight satisfying Equation 5.9. Conversely, if $\tau = 0$, the vertex v_S must have edges whose weights are equal to the weights of edges in v_T, otherwise the mapping $w_S \rightarrow w_T$ will be pruned from the search tree.

To better illustrate this process, consider the degree calculation between vertices T_1 (Figure 25) and S_1 (Figure 26). Notice that the mapping $S_1 \rightarrow T_1$ is viable according to Ullmann's original algorithm if the direction and weights are ignored, as the degree of S_1 (two vertices linked to it) is greater than the grade of T_1 (only one connection). However, to apply the proposed refinement process with $\tau = 1.0$, it is necessary to compare the weight of each edge in T_1 (there is only one edge going out with weight $w_T = 3$) with the edges of S_1 that have the same orientation (there are two edges going out with weights $w_S^1 = 5$ and $w_S^2 = 7$). Next, the result of Equation 5.9 when comparing w_T to w_S^1 is shown:

$$\begin{aligned} | w_T - w_S | &\leq \tau * w_T \\ =| 3 - 5 | &\leq 1.0 * 3 \\ =| -2 | &\leq 3 \\ = 2 &\leq 3 \\ = True. & \end{aligned}$$

However, the comparison between w_T and w_S^2 leads to a distinct result:

$$\begin{aligned} | w_T - w_S | &\leq \tau * w_T \\ =| 3 - 7 | &\leq 1.0 * 3 \\ =| -4 | &\leq 3 \\ = 4 &\leq 3 \\ = False. & \end{aligned}$$

In this case, the grade assigned to S_1 would be "1" because only w_S^1 satisfied the threshold test. Besides, rising the value of τ can increase the grade of S_1, while any reduction may minimize its grade.

In practice, high values of τ lead to fewer optimizations in the search, while lower values will increase the pruning of the search tree. Consequently, there are two contrasting ways to use τ to filter the solutions, either by starting with a low τ (that can lead to an empty search tree) and gradually increase its value in search for some good solution, or by having a high initial value for τ (that generates a massive amount of data to process) and reduce its value continuously until there is a feasible number of solutions to explore. The proposed algorithm uses the latter approach, since the goal is to find the best solution possible by narrowing the search space only when necessary. Thus, the pruning occurs based on the number of solutions to be processed, that must be lower than a specified limiting factor $\mathscr{M}$ to allow for a parallel procedure to select the best solution from the remaining data.

A recurring problem with this implementation is that reductions in the value of τ can prune the search tree so as to have no valid solutions left. To solve this problem, the proposed algorithm stores a matrix containing the last value of τ for each edge, and a matrix to carry information about invalid solutions linked to each edge. Therefore, the method performs a systematic procedure to decrease the value of τ by a factor δ at each step, while monitoring the eradication of solutions. Thus, it is possible to undo any changes made and also flag the edges that no longer can have their τ reduced.

In a nutshell, the PWU can be divided into three main steps. First, it is necessary to create and initialize all the data structures that are needed during execution, including a root matrix M (size $|V_T| \times |V_S|$) that encodes the search tree (Figure 14), a matrix T^τ (size $|V_T| \times |V_T|$) to store the threshold of each edge, and a matrix F^τ (size $|V_T| \times |V_T|$) to flag the edges that can have their corresponding τ reduced.

In the second step, the algorithm enters the pruning phase, where elements of the matrix T^τ have their value decreased gradually to discard unfitting solutions. This step begins by reducing τ for all elements until there are valid solutions, next vertices are selected to be pruned, which consists in reducing τ for specific lines of T^τ, and finally a more accurate refinement is applied to each element of T^τ. By first pruning the entire matrix T^τ and then some lines, the algorithm trims large branches of the search tree before focusing on more careful cases that can be addressed individually.

The third and final step is responsible for selecting the best among the remaining solutions filtered earlier. It uses a parallel approach to compute the minimum difference in game balancing as presented in Section 5.1. Given that in some cases the search tree can give rise to

a massive number of solutions, the parallel algorithm relies on the previous step to receive a maximum volume of solutions to process, that is defined by the constant $\mathscr{M}$. As a result, the second step prunes the search tree repeatedly until the number of mappings is less than $\mathscr{M}$.

In terms of complexity, this algorithm depends mostly on the pruning phase, that reduces the value of elements in matrix T^τ and checks for valid solutions. This process has complexity $O(|V_S|)$, as it is necessary to process each valid candidate. Furthermore, if the threshold defined by $\mathscr{M}$ is satisfied, the algorithm must process each of these solutions, thus presenting complexity $O(\mathscr{M})$. Consequently, the complexity of the PWU algorithm can be defined as $O(N|V_S| + \mathscr{M})$, where N is the number of iterations the algorithm performs when pruning the search tree.

Algorithm 5 provides a high level description of this process in pseudocode, and the next section exemplifies this approach using two simple graphs.

5.3.1 Example

Consider the game models shown G_T and G_S used in Section 5.2.1. Moreover, let's define $\tau = 2.0$, $\delta = 0.5$ and $\mathscr{M} = 5$ as initial parameters to the algorithm. In this case, the method defines M, T^τ and F^τ as follows:

$$M = \begin{vmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{vmatrix}, T^\tau = \begin{vmatrix} 2.0 & 2.0 & 2.0 \\ 2.0 & 2.0 & 2.0 \\ 2.0 & 2.0 & 2.0 \end{vmatrix}, F^\tau = \begin{vmatrix} True & True & True \\ True & True & True \\ True & True & True \end{vmatrix}$$

The initial configuration of M includes all possible solutions, so each vertex $v_T \in V_T$ can be mapped to any vertex $v_S \in V_S$, totaling ${}^{|V_S|}P_{|V_T|} = \frac{|V_S|!}{(|V_S - V_T|)!}$ (24 solutions). Since the number of solutions is bigger than $\mathscr{M}$, the Equation 5.9 is used to validate the mappings. To illustrate the process, one should focus on the steps that associate vertex T_1 to vertex S_1 as detailed below (Table 4). In this case, T_1's only edge $e_{T_1 \to T_2}$ must have its weight matching the weight of at least one candidate edge in S_1 (edges $e_{S_1 \to S_2}$ and $e_{S_1 \to S_4}$).

Since both edges $e_{S_1 \to S_2}$ and $e_{S_1 \to S_4}$ can be mapped to $e_{T_1 \to T_2}$ when using $\tau = 2.0$, there is no change to the corresponding element of the root matrix M. Consequently, the number of possible solutions remains the same, and the algorithm must decrease the corresponding threshold in T^τ ($\tau_{T_1 \to S_1}$) to prune the search tree. Since $\delta = 0.5$, thresholds are halved and the

Algorithm 5: Algorithm describing the Parallel Weighted Ullmann.

Input: Matrix A_T, Matrix A_S, Initial τ, Factor δ, Max number of solutions to process $\mathscr{M}$

Output: A matrix P_R that encodes the best solution found

```
begin
    Create the root matrix M to encode the search tree;
    Create a matrix T^τ with the dimensions of A_T;
    Initialize the elements of T^τ with the value τ;
    Create a matrix F^τ with the dimensions of A_T;
    Initialize the elements of F^τ with the value True;
    while 𝓜 < solutions in M do
        for w_ij ∈ A_T do
            if F_ij^τ is True then
                Reduce the value of T_ij^τ by the factor δ;
                if Has no valid solutions with T^τ then
                    Restore the last value of T_ij^τ;
                    F_ij^τ = False;
                else
                    Update solutions in M;
                end
            end
        end
    end
    P_R=Select the best solution in M;
end
```

Table 4 – Table presenting the checking of Equation 5.9 between T_1 and S_1 when $\tau = 2.0$.

Edge $e_{S_1 \to S_2}$	Edge $e_{S_1 \to S_4}$
$\mid w_{T_1 \to T_2} - w_{S_1 \to S_2} \mid \leq \tau * w_{T_1 \to T_2}$	$\mid w_{T_1 \to T_2} - w_{S_1 \to S_4} \mid \leq \tau * w_{T_1 \to T_2}$
$= \mid 3-5 \mid \leq 2.0*3$	$= \mid 3-7 \mid \leq 2.0*3$
$= \mid -2 \mid \leq 6$	$= \mid -4 \mid \leq 6$
$= 2 \leq 6$	$= 4 \leq 6$
$= True$	$= True$

Source: Author

algorithm applies reductions to the entire matrix T^τ as follows:

$$T^\tau = \begin{vmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{vmatrix}$$

Again, the math behind the mapping between T_1 and S_1 with $\tau_{T_1 \to S_1} = 1.0$ is presented, as shown in Table 5.

Table 5 – Table presenting the checking of Equation 5.9 between T_1 and S_1 when $\tau_{T_1 \to S_1} = 1.0$.

Edge $e_{S_1 \to S_2}$	Edge $e_{S_1 \to S_4}$
$\mid w_{T_1 \to T_2} - w_{S_1 \to S_2} \mid \leq \tau_{T_1 \to S_1} * w_{T_1 \to T_2}$ $= \mid 3 - 5 \mid \leq 1.0 * 3$ $= \mid -2 \mid \leq 3$ $= 2 \leq 3$ $= True$	$\mid w_{T_1 \to T_2} - w_{S_1 \to S_4} \mid \leq \tau_{T_1 \to S_1} * w_{T_1 \to T_2}$ $= \mid 3 - 7 \mid \leq 1.0 * 3$ $= \mid -4 \mid \leq 3$ $= 4 \leq 3$ $= False$

Source: Author

Notice that the mapping between edges $e_{T_1 \to T_2}$ and $e_{S_1 \to S_4}$ is no longer valid for $\tau_{T_1 \to S_1} = 1.0$, therefore consider that T_1 could only be mapped to S_1, then the vertex T_2 could never be mapped to S_4. To represent this instance, the root matrix must be:

$$M = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{vmatrix}$$

This configuration of M indicates that T_1 is mapped to S_1 (a one in the first line and column of M), meanwhile, the second line specifies that T_2 can be mapped to S_2 or S_3. Naturally, further reductions in $\tau_{T_1 \to S_1}$ can lead to more prunes, and even invalid solutions.

For instance, if the method sets $\tau_{T_1 \to S_1} = 0.5$, the mapping between T_1 and S_1 is validated according to the operations shown in Table 6. In this case, if S_1 is the last remaining vertex that T_1 could be assigned to, the matrix M would have zeros in all elements of its first line, thus violating the bijective mapping defined by Equation 5.1. Next, to avoid this situation, the algorithm undoes the last reduction in $\tau_{T_1 \to S_1}$ and alters F^τ to indicate that $\tau_{T_1 \to S_1}$ has reached its minimum value. This process is depicted by the matrices below:

Before reducing the value of $\tau_{T_1 \to S_1}$:

$$M = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{vmatrix}, T^\tau = \begin{vmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{vmatrix}, F^\tau = \begin{vmatrix} True & True & True \\ True & True & True \\ True & True & True \end{vmatrix}$$

During reduction matrix M becomes invalid:

$$M = \begin{vmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{vmatrix}, T^\tau = \begin{vmatrix} 0.5 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{vmatrix}, F^\tau = \begin{vmatrix} True & True & True \\ True & True & True \\ True & True & True \end{vmatrix}$$

After undoing the reduction and signaling in F^{τ} that $\tau_{T_1 \to S_1}$ must not be reduced:

$$M = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{vmatrix}, T^{\tau} = \begin{vmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{vmatrix}, F^{\tau} = \begin{vmatrix} False & True & True \\ True & True & True \\ True & True & True \end{vmatrix}$$

Table 6 – Table presenting the checking of Equation 5.9 between T_1 and S_1 when $\tau = 0.5$.

Edge $e_{S_1 \to S_2}$	Edge $e_{S_1 \to S_4}$
$\mid w_{T_1 \to T_2} - w_{S_1 \to S_2} \mid \leq \tau_{T_1 \to S_1} * w_{T_1 \to T_2}$ $= \mid 3 - 5 \mid \leq 0.5 * 3$ $= \mid -2 \mid \leq 1.5$ $= 2 \leq 1.5$ $= False$	$\mid w_{T_1 \to T_2} - w_{S_1 \to S_4} \mid \leq \tau_{T_1 \to S_1} * w_{T_1 \to T_2}$ $= \mid 3 - 7 \mid \leq 0.5 * 3$ $= \mid -4 \mid \leq 1.5$ $= 4 \leq 1.5$ $= False$

Source: Author

The reduction in the elements of T^{τ} is repeated until the number of solutions is less than $\mathscr{M}$ or all elements of F^{τ} are set to $False$. In the latter case, the PWU fails to execute, because if all elements of F^{τ} are $False$ it indicates additional reductions in T^{τ} leads to invalid configurations of M. Besides, the amount of data to process is still too great for a deterministic approach to handle. This situation can occur depending on the size of the game model and the search space. To address this situation, the next section details a non-deterministic approach that is capable of managing massive search spaces.

5.4 Genetic Algorithm

This work proposes the use of PCG to transpose maps of LBGs, regardless of their size and features. Since the subgraph isomorphism problem grows exponentially, in some cases a non-deterministic method is required to address the challenge.

This section presents a non-deterministic algorithm implemented to tackle instances of the graph isomorphism problem that are unfeasible to be processed by the previous algorithms. The proposed approach is based on (LI *et al.*, 2016), that explores the subgraph isomorphism problem using three evolutionary methods: Simulated Annealing, (1+1) evolutionary algorithm, and Genetic Algorithm (GA). Li *et al.* (2016) concluded that the GA delivered better results than the other algorithms in most cases. Therefore, in this work, a GA was adapted to process weighted graphs and use $\mathfrak{M}$ as objective function.

There are numerous concepts and variations regarding the implementation of GAs, and the key definitions necessary to understand the proposed method are provided here. First,

GAs are based on processes observed in nature, such as natural selection, mutation, crossover, etc. The goal is to use some criteria to select the best individual (also known as genotype) within a population of chromosomes. In this case, given two graphs $G_T = (V_T, E_T, W_T)$ and $G_S = (V_S, E_S, W_S)$, the mapping between a pair of vertices (v_T/v_S) of the graphs is called gene, and a complete bijective mapping $(\mathscr{F} : V_T \rightarrow V_S)$ is referred to as chromosome. Accordingly, all the possible combinations of solutions make up the entire chromosome population.

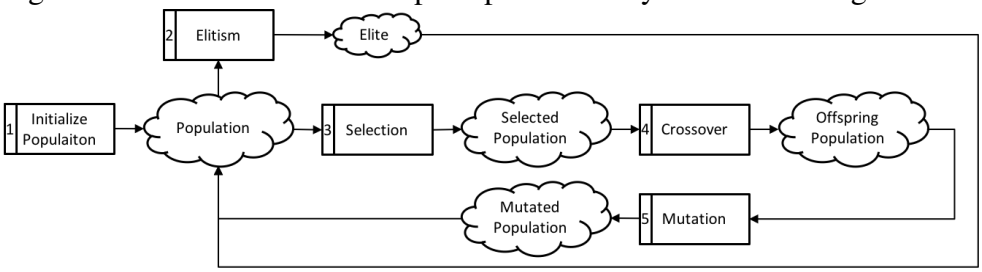
The algorithm starts by randomly selecting an initial population to be evolved during execution (often a percentage ψ of the total population). Similar to MCTS, the GA has its execution restricted by a certain budget, which in this work is the number of generations η to evolve the initial population. In every generation, the population undergoes phases akin to biological evolutionary processes. Usually, developing a GA consists in adapting the problem one wants to solve to these concepts, and implementing a set of computational steps that were proposed to mimic processes that occur spontaneously as part of natural selection.

The purpose is to create a better generation of chromosomes (solutions) that are assessed according to a specific criterion called fitness function. As discussed in Section 5.1, the balanced transposition of maps of LBGs was formulated as an optimization problem whose goal is to minimize $\mathfrak{T}$, hence the developed GA uses the same optimization as fitness function.

In general, the sequence of steps implemented to spawn a new generation of chromosomes is *Selection*, *Crossover*, and *Mutation*. Besides, the literature contains information about variations and improvements regarding each of these steps, depending on the purpose of the algorithm and the problem to be solved.

In this work, the GA included an additional feature called *Elitism*, that consists in maintaining a percentage ε of the best fit chromosomes in the next generation, as illustrated in Figure 33. This step ensures future generations will contain at least the best solutions from past executions, because both *Crossover* and *Mutation* are not guaranteed to always produce better results.

Figure 33 – Overview of the steps implemented by the Genetic Algorithm.



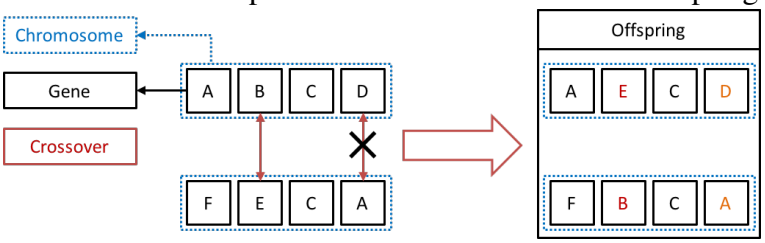
Source: Author

Selection is a step responsible to appoint the chromosomes that will exchange genes later in the *Crossover* phase. It can be performed in many ways, such as reward-based, truncation, fitness proportionate, tournament, among others. This work uses a deterministic tournament selection for being a fast method that avoids minimum locals, since it does not focus on keeping particular traits (genes) nor it is rewarded by the quality of the offspring. Tournament selection starts by randomly selecting chromosomes from the population, and then assessing their fitness to decide the winner. In this case, chromosomes that win their respective tournaments are matched to undergo crossover and generate a new solution (also known as offspring). Usually, the amount of chromosomes selected to participate in tournaments is defined by a percentage ρ.

The process called *Crossover* works analogously to the homonym activity in the nucleus of cells. Once two chromosomes are selected, they can exchange some of their genes. There are many types of crossover techniques to be used in GAs, such as single-point, two-point, k-point, uniform crossover, partial-mapped crossover, order one crossover, alternating-position crossover, among others. However, the single-point, two-point and k-point crossover select one or more regions of the chromosome and trade entire portions of their genes, thus they are prone to generate invalid solutions. For instance, there cannot be the same vertex of one graph mapped to two distinct vertices of the other, given that the subgraph isomorphism requires the creation of a bijective mapping ($\mathscr{F} : V_T \rightarrow V_S$). Crossover operators such as partial-mapped, order one, and alternating-position crossover always generate valid solutions for permutations, but they require additional checking to avoid violating the bijective mapping of the graph isomorphism. In practice, using the uniform crossover has shown a better compromise between the quality of solutions and performance, since it has faster execution and operates by selecting unique genes that can be quickly checked if are already present in the chromosome counterpart before being exchanged. Figure 34 illustrates this process showing an invalid offspring that has two distinct genes with the same content, while the other is correct. In this case, the permutation between the last genes (“A” and “D”) is prevented to not generate unsuitable solutions.

The last step of a GA is *Mutation*, which can be understood as a random change in the genes of a chromosome. Its main purpose is to introduce diversity in the population, thus ensuring the algorithm will explore otherwise unexpected solutions. To implement this step, it is necessary to define the probability μ of a chromosome to suffer a mutation. In general, the higher μ is, the GA becomes more exploratory. So if $\mu = 1.0$ every chromosome in the population will experience mutation, thus making the GA to behave like a random search.

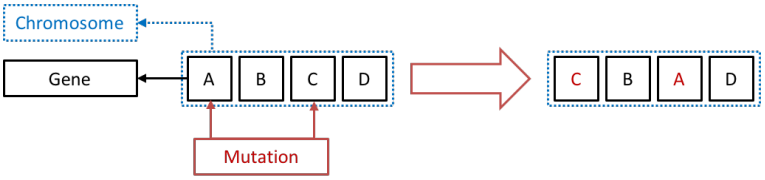
Figure 34 – Example of uniform crossover where the output is composed of a valid and an invalid offspring.



Source: Author

Furthermore, there are distinct types of mutation, such as Random Resetting, Scramble, Swap, Inversion, etc. The GA implemented in this work makes use of the swap mutation, as it selects genes from the chromosome itself to switch positions (Figure 35), thus always generating valid solutions.

Figure 35 – Figure depicting the swap mutation implemented in this work.



Source: Author

All the aforementioned steps are executed repeatedly for a certain number of generations $O(\eta)$. Thus, the resulting complexity is $O(\eta(\psi \log_{\psi} + \rho\psi + \mu\psi))$, as the algorithm repeats the sorting and selection of chromosomes $O(\psi \log_{\psi})$, the amount of elements to undergo tournament $O(\rho\psi)$, and the mutation rate $O(\mu\psi)$. Consequently, unlike the PWU and the MCTS, the performance of the GA is invariant to the size of graphs G_T and G_S.

Algorithm 6 describes key aspects of the GA implemented in this work. Next, an example showcases how the algorithm operates.

5.4.1 Example

Similarly to sections 5.3.1 and 5.2.1, the game models G_T (Figure 25) and G_S (Figure 26) are used to showcase how the GA works. For the sake of simplicity, let's define the following parameters to the algorithm: Population size($\psi = 8$), Number of generations($\eta = 1$), Elitism factor($\varepsilon = 0.25$), Tournament size($\rho = 0.5$), and Mutation probability($\mu = 0.5$).

First, the algorithm must randomly create an initial population with size 8 (ψ). Figure 36 presents some chromosomes depicting the mapping between vertices of G_S and G_T to

Algorithm 6: Algorithm describing the Genetic Algorithm.

Input: Matrix A_T, Matrix A_S, Population size ψ, Number of Generations η, Elitism factor ε, Tournament size ρ, Mutation probability μ

Output: A matrix P_R that encodes the best solution found

begin

  Randomly generate an initial population from A_S with size ψ;

  while *Number of generations* $< \eta$ **do**

    Select the best $(\varepsilon * \psi)$ chromosomes from the population to form an elite;

    Select $(\rho * \psi)$ chromosomes to undergo tournament selection;

    Perform crossover between winners of the tournament;

    Apply mutation to the resulting population according to the probability μ;

    Combine the elite and the offspring to form a new population;

    Select the best ψ elements from this new population that optimizes $\mathfrak{M}$;

  end

  Build matrix P_R from the best solution in the resulting population;

end

be used in this example. Notice that this initial population can also contain invalid solutions (e.g., chromosome "b" originates an unfeasible solution because G_S does not have edges connecting vertices S_2 and S_4).

Figure 36 – Chromosomes illustrating an initial population with size $\psi = 8$.

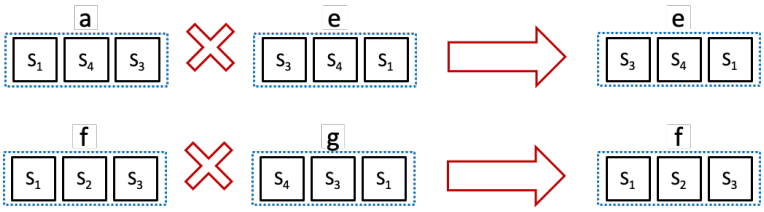
Source: Author

Assuming the initial population is created, the next phase builds an elite. This step requires sorting each chromosome in the population by their $\mathfrak{M}$ and then selecting the first ones. The current example results in the selection of two chromosomes $(\varepsilon * \psi = 0.25 * 8 = 2)$ to be part of the elite. The table below shows the values of $\mathfrak{M}$ for each chromosome in Figure 36.

c	d	e	f	h	a	b	g
$\mathfrak{M} = 4$	$\mathfrak{M} = 5$	$\mathfrak{M} = 5$	$\mathfrak{M} = 6$	$\mathfrak{M} = 6$	$\mathfrak{M} = 9$	*Invalid*	*Invalid*

In this case, the chromosomes “c” and “d” enter the elite population, and the algorithm moves to the selection phase. Now, the method randomly chooses four ($\rho * \psi = 0.5 * 8 = 4$) chromosomes to undergo tournament selection. Figure 37 depicts this process, assuming that the pairs of chromosomes (a,e) and (f,g) were chosen. In this case, chromosomes “e” and “f” were the winners because they minimize $\mathfrak{M}$.

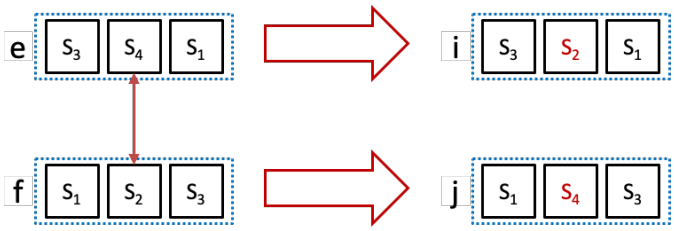
Figure 37 – Depiction of a tournament selection.



Source: Author

Next, the chromosomes selected by tournament are matched to undergo crossover, so the algorithm randomly chooses genes to be exchanged between them. The amount of genes to be exchanged varies stochastically. Figure 38 shows a crossover that includes the second gene of the chromosomes.

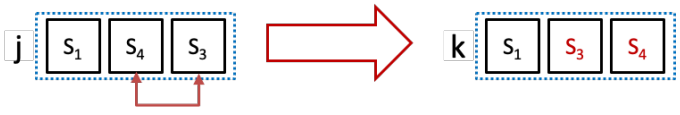
Figure 38 – Uniform crossover exchanges specific genes between chromosomes.



Source: Author

The last step applies mutation to the remaining population. Thus the method chooses a chromosomes according to a specific probability. In this example, half of the population is likely to be subjected to this process ($\mu = 0.5$). Once a chromosome is chosen, the algorithm randomly selects a pair of its genes to be swapped, as illustrated in Figure 39.

Figure 39 – Depiction of mutation being executed in one of the chromosomes.



Source: Author

Finally, the method combines all the populations, including the elite, and the fittest $\psi = 8$ chromosomes are selected to compose the new generation. In this case, the table below shows the resulting calculus of $\mathfrak{M}$ for the resulting population, hence the method must select chromosomes “i”, “c”, “d”, “e”, “f”, “h”, “a”, and “j” to repeat the process. However, the example defines the number of generations as one ($\eta = 1$). As a result, the algorithm selects the best solution (chromosome “i”) and returns it as the result.

i	c	d	e	f	h	a	j	b	g	k
$\mathfrak{M} = 2$	$\mathfrak{M} = 4$	$\mathfrak{M} = 5$	$\mathfrak{M} = 5$	$\mathfrak{M} = 6$	$\mathfrak{M} = 6$	$\mathfrak{M} = 9$	$\mathfrak{M} = 9$	*Invalid*	*Invalid*	*Invalid*

5.5 Conclusion

In a nutshell, this chapter showed how the challenge of transposing maps of LBGs can be modeled and addressed as a WGMP. First, a formulation of the problem was presented in Section 5.1. It provides details on how to handle the many possible mappings between graphs and defines the challenge as an optimization problem that can be solved by minimizing $\mathfrak{M}$.

Next, three methods to tackle the WGMP were presented. An algorithm based on the MCTS (Section 5.2) method that has been previously used in (MAIA *et al.*, 2017) and (FERREIRA *et al.*, 2019), a novel deterministic approach that builds on the well known Ullmann’s algorithm, called Parallel Weighted Ullmann (Section 5.3), and a Genetic Algorithm (Section 5.4). The key concepts behind each algorithm were presented, and their implementation was detailed along with simple examples showcasing the steps performed by each method during execution.

6 EVALUATION

This chapter describes an evaluation conducted to assess the algorithms detailed in the previous chapter. The two main aspects to be measured are the quality of results, that can be gauged using $\mathfrak{M}$, and the time each algorithm takes to compute the solutions.

It is important to highlight that, although these algorithms can potentially solve the WGMP in other applications, they were conceived to address the map transposition challenge, hence their main focus is to find good solutions for graphs with limited size as fast as possible. As a result, they may not operate well with large and complex graphs that are common in other fields, such as data mining, biological interaction, etc.

The evaluation is divided into two main groups (A and B). The first group (A) comprises game models with smaller sizes, depicted by graphs containing 5 and 10 vertices. Games with up to 10 POIs can be successfully processed by deterministic methods, so all three algorithms (PWU, MCTS, and GA) and a Parallel Brute-Force (PBF) approach were benchmarked with these graphs. The PBF method outputs the optimum solution, thus providing a reference to the results generated by the other algorithms, and can also be used to showcase the efficiency of the pruning proposed by the PWU method. However, likewise the dynamic programming approach presented in (MACVEAN *et al.*, 2011), the PBF approach is not eligible to transpose game maps on demand due to their long processing time.

The second group (B) is composed of game models whose corresponding graphs contains 20 and 30 vertices, so the amount of possible solutions to be searched increases drastically. Consequently, they must be handled by non-deterministic methods (MCTS and GA).

The chapter is divided into six sections. Section 6.1 details the material and methods used in this evaluation, then sections 6.2 and 6.3 presents the quality and time generated from trials, respectively. Section 6.4 discusses the results, and Section 6.5 mentions threats to this evaluation. Finally, Section 6.6 closes the chapter.

6.1 Materials and Methods

The algorithms were mostly implemented using the *Python* programming language, except for parts of the PWU and the PBF that relied on parallelism and were coded using the *Open Computing Language* (OpenCL)[1]. In this case, the main program is still executed in

[1] https://www.khronos.org/opencl/

python, and only some tasks run in parallel. The main advantage of this approach is that OpenCL automatically manages threads in processors and Graphics Processing Units (GPUs), thus acting as a layer between the main program and the hardware in charge of running the parallel code. The kernels used in this implementation are available in the Appendix B.

The tests were conducted on a *c4.2xlarge* machine with an 8 core Intel Xeon E5-2666 CPU and 15GB of memory, hosted on Amazon Cloud. This machine has great processing resources to be explored by OpenCL to execute the PWU.

Given that the proposed approach relies on webservices to provide information about the locations the maps will be transposed to, and that these APIs generally provide data in the shape of an adjacency matrix of a complete graph, the evaluation generated complete graphs randomly to emulate real inputs to the algorithms. The weights of graphs were set to be in the range $[0-100]$. In addition, to symbolize the original game map (G_T), graphs with size 5, 10, 20, and 30 ($|V_T|$) were used, and graphs with sizes 20, 30, 40, and 50 ($|V_S|$) represented the search space (G_S), according to the formulation presented in Section 5.1.

As mentioned earlier, tests were divided into two groups (A and B) due to the great difference in complexity regarding the number of solutions to explore. Consequently, the parameters used to configure the algorithms changed for each group. Table 7 shows the settings used by the algorithms during trials with group A, and Table 8 presents the parameters defined with the group B.

Table 7 – Parameters used to configure the algorithms for tests with the group A.

MCTS	$\kappa = 0.05, \mathscr{B} = 5 \times 10^3$
PWU	$\tau = 1.0$, $\delta = 0.5$, $\mathscr{M} = 2.5 \times 10^6$
GA	$\eta = 10$, $\varepsilon = 0.1$, $\rho = 10$, $\mu = 0.5$, $\psi = 7.5 \times 10^3$

Source: Author

Table 8 – Parameters used to configure the algorithms for tests with the group B.

MCTS	$\kappa = 0.05, \mathscr{B} = 1 \times 10^4$
GA	$\eta = 10$, $\varepsilon = 0.1$, $\rho = 10$, $\mu = 0.5$, $\psi = 2.5 \times 10^3$

Source: Author

The evaluation consisted in applying the transposition algorithms to 30 randomly generated pairs of graphs (G_T,G_S). The final result was given by the average $\mathfrak{M}$ and the turnaround time gauged for each pair. Next, the results were grouped by the size of G_T to analyze the performance of each algorithm when the same game model is transposed to search spaces with distinct sizes.

6.2 Quality Results

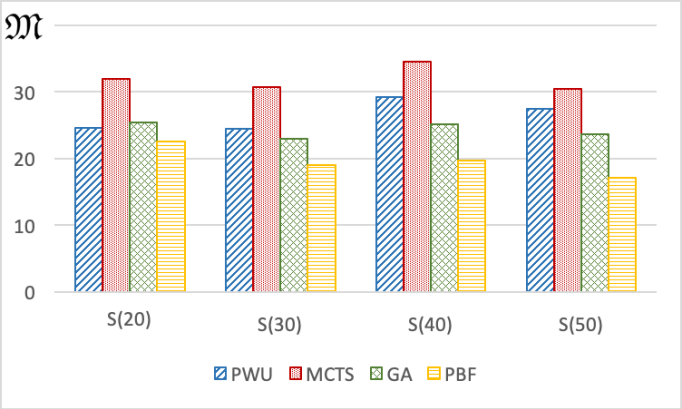
As previously mentioned, the quality of the transposition can be measured by $\mathfrak{M}$. First, group A was analyzed, detailing the experiments for graphs G_T with size 5 and 10. In the former case, the results showed that GA and PWU operate similarly, especially for smaller graphs G_S. Both methods generate outputs close to the optimum solution (PBF), while MCTS produces results with higher $\mathfrak{M}$. Table 9 details the resulting values of $\mathfrak{M}$ and Figure 40 depicts the traits aforementioned.

Table 9 – Table showing the average of $\mathfrak{M}$ for graphs G_T with size 5.

$\lvert V_T\rvert = 5$	PWU	MCTS	GA	PBF
$\lvert V_S\rvert = 20$	24.54	31.90	25.39	22.56
$\lvert V_S\rvert = 30$	24,46	30.76	22.95	19.01
$\lvert V_S\rvert = 40$	29.16	34.46	25.16	19,74
$\lvert V_S\rvert = 50$	27.45	30.40	23,67	17,10

Source: Author

Figure 40 – Chart depicting the average $\mathfrak{M}$ for different sizes of G_S and $\lvert V_T\rvert = 5$.



Source: Author

For graphs G_T with size 10, the PBF was no longer used, because the time to process is impracticable, as shown in the next section. Thus, Table 10 presents the average $\mathfrak{M}$ for PWU, MCTS, and GA. In this case, the algorithms delivered very similar results, as MCTS and PWU present virtually equal results, and GA shows minor improvements over the others.

Furthermore, variations in the size of the search space (G_S) have not produced great impacts in reducing $\mathfrak{M}$. Figure 41 depicts this trait clearly, and highlights the slightly better results achieved by the GA.

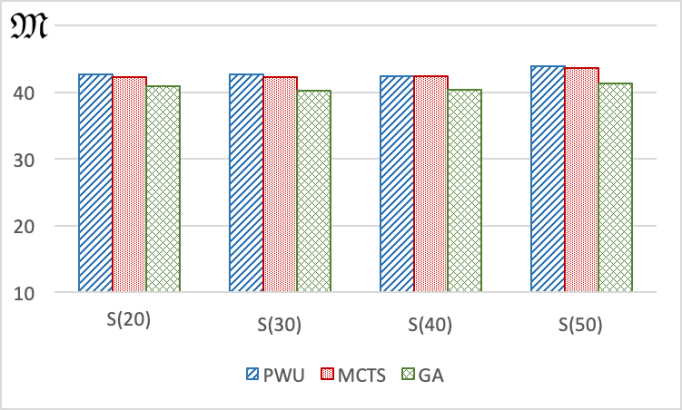
For group B, graphs depicting game models composed of 20 and 30 POIs were used. In this case, PWU fails in the majority of cases, either by not pruning the search tree enough to

Table 10 – Average $\mathfrak{M}$ for $|V_T| = 10$ and varying $|V_S|$.

$\|V_T\| = 10$	PWU	MCTS	GA
$\|V_S\| = 20$	42.66	42.23	40.94
$\|V_S\| = 30$	42.65	42.25	40.29
$\|V_S\| = 40$	42.43	42.45	40.41
$\|V_S\| = 50$	43.96	43.62	41.32

Source: Author

Figure 41 – Chart depicting the average $\mathfrak{M}$ for different sizes of G_S and $|V_T| = 10$.



Source: Author

make it feasible to search, or by eliminating so many branches that no valid solutions are left. As expected, this challenge must be tackled by non-deterministic methods. Consequently, MCTS and GA are suited to this task. However, the last step in the MCTS approach looks for the best automorphism of a solution (Algorithm 4). This feature was disabled in this trial due to the large amount of solutions to process.

Table 11 shows the data generated using these methods. MCTS has shown poor results while GA has managed to output satisfying solutions. It is clear that MCTS did not converge to more promising solutions. Even after increasing its budget, as shown in Table 8, the method selects random solutions, thus yielding unsatisfying solutions.

Table 11 – Average $\mathfrak{M}$ for $|V_T| = 20$ and varying $|V_S|$.

$\|V_T\| = 20$	MCTS	GA
$\|V_S\| = 20$	10809.4	55.60
$\|V_S\| = 30$	10935.8	54.33
$\|V_S\| = 40$	10948.3	54.57
$\|V_S\| = 50$	10876.8	54.15

Source: Author

Lastly, considering that MCTS cannot output satisfying solutions when $|V_T|$ increases, results were produced applying the GA when $|V_T| = 30$. Table 12 shows both $\mathfrak{M}$ and time measured during the tests.

Table 12 – Average $\mathfrak{M}$ using GA for $|V_T| = 30$.

$\|V_T\| = 30$	$\mathfrak{M}$
$\|V_S\| = 30$	58.31
$\|V_S\| = 40$	58.93
$\|V_S\| = 50$	57.87

Source: Author

In general, the bigger graphs G_T and G_S are, the higher is the $\mathfrak{M}$ of solutions generated by the methods. However, even though the number of solutions grows exponentially with size, the GA has proven to be reliable in finding acceptable solutions in a relatively short time. The next section presents the performance results for both groups A and B.

6.3 Performance Results

Regarding processing time, the group A obviously performs better than the group B, as the search space is significantly smaller. For graphs G_T with size 5, it is clear that PWU and GA have similar performance, with PWU being faster on average, as shown in Table 13. Moreover, the processing time of MCTS increases with the size of G_S, as the partial tree built to explore solutions also grows. This trend can be observed in the next trials as well. Meanwhile, the time required by PBF to compute solutions grows drastically, thus becoming an unfeasible algorithm to be used on demand, as shown in Figure 42.

Table 13 – Table showing the average turnaround time in seconds for graphs G_T with size 5.

$\|V_T\| = 5$	PWU	MCTS	GA	PBF
$\|V_S\| = 20$	1.00	0.77	0.88	0.73
$\|V_S\| = 30$	0.78	1.06	1.07	7.17
$\|V_S\| = 40$	0.89	1.79	1.07	36.77
$\|V_S\| = 50$	1.33	2.73	1.10	131.09

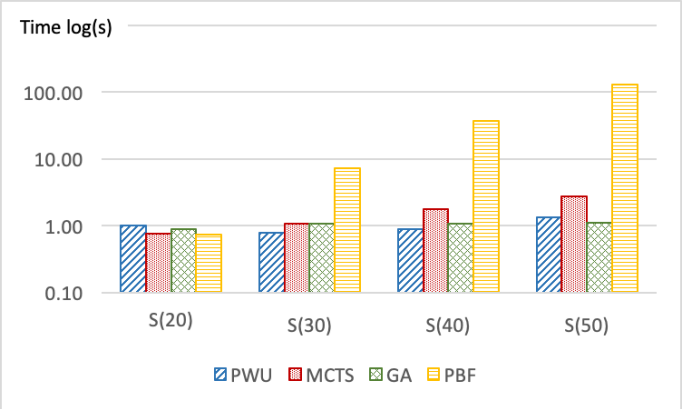
Source: Author

For graphs G_T with size 10, the time used by each algorithm is quite different. While GA takes a steady amount of time to find solutions regardless of the search space, both MCTS and PWU are sensitive to the variations in the size of G_S. In general, the bigger the search space is, the slower both methods are. However, PWU has yielded faster performance in all scenarios, as shown in Table 14.

It is also clear that the time required by MCTS increases directly with higher $|V_S|$, thus corroborating to the trend mentioned in the beginning of this section. Figure 43 provides a good illustration of this trait.

In the case of larger graphs, such as the ones from the group B, only MCTS and GA

Figure 42 – Average time each algorithm used to process G_T with size 5, and varying $|V_S|$.



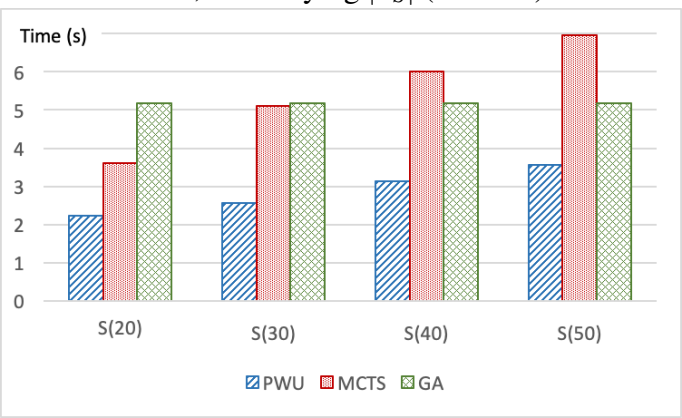
Source: Author

Table 14 – Table showing the average turnaround time in seconds when $|V_T| = 10$.

$\|V_T\| = 10$	PWU	MCTS	GA
$\|V_S\| = 20$	2.24	3.61	5.17
$\|V_S\| = 30$	2.57	5.10	5.17
$\|V_S\| = 40$	3.13	6.02	5.17
$\|V_S\| = 50$	3.56	6.97	5.18

Source: Author

Figure 43 – Average time to process graphs with $V_T = 10$, and varying $|V_S|$ (seconds).



Source: Author

are eligible to address the task. However, while the performance of the former increases with the size of G_S, the GA has delivered a steady and reliable performance, as shown in Table 15. As expected, MCTS requires even more time to compute solutions, so as to become unsuitable to handle bigger graphs, as depicted in Figure 44.

Finally, Table 16 shows that the performance of the GA is invariant to the size of G_S, thus making it a stable approach to be used in larger search spaces.

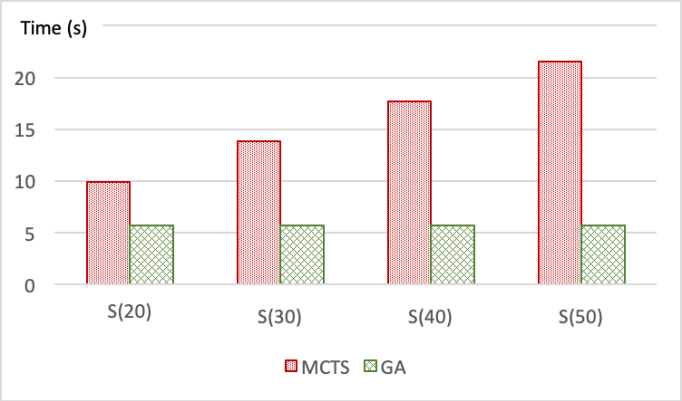
In summary, it is clear that all methods have their performance directly linked to the size of G_S. However, while for graphs from the group A, the PWU revealed to be the fastest, the

Table 15 – Average time in seconds for $|V_T| = 20$.

$\|V_T\| = 20$	MCTS	GA
$\|V_S\| = 20$	9.91	5.71
$\|V_S\| = 30$	13.87	5.72
$\|V_S\| = 40$	17.69	5.70
$\|V_S\| = 50$	21.55	5.69

Source: Author

Figure 44 – Average time in seconds algorithms used to process graphs with $V_T = 20$.



Source: Author

Table 16 – Average time in seconds using GA for $|V_T| = 30$.

$\|V_T\| = 30$	Time(s)
$\|V_S\| = 30$	13.38
$\|V_S\| = 40$	13.37
$\|V_S\| = 50$	13.37

Source: Author

GA is the unique approach capable of handling graphs from the group B regardless of changes in the size of G_S. The next section discusses the results presented in both sections 6.2 and 6.3.

6.4 Discussion

Results shown in the previous sections indicate all methods deliver good performance for games that have up to 10 POIs. In these cases, the algorithms are able to find solutions close to the optimal, but spending only a fraction of the time a parallel brute-force method takes.

In general, PWU is faster, while GA produces slightly better solutions. Although MCTS lags behind both methods, it still shows a good compromise between time and quality. Thus, all algorithms can be used to transpose maps of LBGs on demand.

Nevertheless, for larger games the amount of solutions quickly explodes, hence only non-deterministic approaches are suited to cope with such a massive search space. Thus, the GA and MCTS were tested, but only the GA has proven to find viable solutions. In fact, the MCTS

is more effective in the long run as the algorithm must recognize the search space in order to focus on better solutions. Given that the number of branches in the search tree is huge, there must have a proportionally big sum of visitations for the algorithm to build a partial tree capable of indicating branches with the best solutions, thus rendering MCTS unsuitable for the task.

A common trend in all tests is that the size of G_S does not impact performance as heavily as variations in the size of G_T. This is inherent to the formula that defines the size of the search space (Equation 5.7). In the case of GA, it is clear that the size of G_T has been the main variable to impact performance since the time to compute solutions when G_S increases hardly changes. This situation happens because the method only holds data about a fixed amount of solutions (population), while both MCTS and PWU have to handle dynamic search trees that vary in size according to the graphs provided as input.

The results shown during tests are promising, especially for LBGs composed of up to 10 POIs. These games can be designed by small or independent studios using authoring tools, as shown in (SILVA *et al.*, 2017) and (FERREIRA *et al.*, 2019).

6.5 Threats to Validity

A key threat to this evaluation regards the graphs used in the tests. During tests 750 graphs were randomly created to mimic the data provided by external APIs, however, it is common for graphs provided by these webservices to have some patterns. For instance, the distance to walk between points A and B may be the same as walking between B and A, thus incurring in two edges with the same length in strategic positions of the graph. These ordinary traits are inherent to each region, thus being hard to predict or emulate. Besides, it is not clear whether some areas of the world can give rise to maps containing unexpected patterns and how the algorithm will behave in such cases. Consequently, an evaluation with users and real data was needed, and is presented in Chapter 7.

Another important aspect to highlight regards the use of parallelism and OpenCL. Since the hardware plays a key role in the execution of parallel methods, it is likely that the resulting turnaround time of PWU and PBF will vary if a powerful GPU or a processor with less cores are used to run the same tests.

6.6 Conclusion

In this chapter tests regarding processing time and quality ($\mathfrak{M}$) were presented. The data were divided into two groups according to the size of graphs used as input. The hardware and parameters used to run the trials and to configure the algorithms were introduced in Section 6.1.

Section 6.3 details the data collected during the tests, according to the size of the graphs used. Moreover, the algorithms were compared regarding the time to find solutions and their quality. Next, Section 6.4 discussed the results and highlighted key aspects of each algorithm. In the last section, threats to the evaluation were introduced, including the need to conduct tests with real data, that will be presented in the next chapter.

7 USER EVALUATION

This chapter presents an evaluation conducted with users to assess the balanced transposition of maps for LBGs. The main purpose of this evaluation is to check whether the proposed approach works appropriately in multiple places despite the style and size of the LBG.

To achieve this goal, four LBGs were developed, each containing distinct characteristics regarding mechanics, the number of places to visit (size), strategy, and game flow. Therefore, it is possible to assess the resiliency of the proposed method to changes in these attributes.

During the test users were asked to analyze the designed LBGs and compare their original game map to a set of game maps generated by distinct transposition algorithms. Then, a statistical investigation was conducted to determine whether $\mathfrak{M}$ (and hence the outcome of each algorithm) is linked to transpositions deemed successful. Furthermore, both $\mathfrak{I}$ and $\sigma_{\mathfrak{I}}$ of the transposed maps were calculated and compared to the original game.

The chapter is organized as follows: Section 7.1 presents the instruments and methods, including the games designed to be used in this evaluation, Section 7.2 introduces the profile of participants, in Section 7.3 the evaluation procedure is detailed, Section 7.4 presents the outcome of trials and a statistical analysis of the data collected, the results are discussed in Section 7.5, Section 7.6 mentions threats to this evaluation, and lastly, Section 7.7 concludes the chapter.

7.1 Materials and Methods

Before detailing the procedure elaborated to evaluate the approach, this section first introduces the material and methods that were used during the trials. In the initial stage it was necessary to design a set of LBGs with varying traits to be assessed. In this case, four games were created, namely *Faith Quest*, *Exploranium*, *Komandant*, and *Impetus*. Later, sections 7.1.2, 7.1.1, 7.1.3, and 7.1.4 detail each game and their respective traits.

In addition, a website was built to present the research and its purpose[1]. It contained the definition of LBGs, mentioned some examples, and discussed the challenges involving game balancing and the transposition issues. Moreover, an animated video[2] was added to illustrate how the proposed approach addresses the problem. Appendix B contains screenshots of each web page presented to users. These instructions were provided before collecting any information from

[1] http://www.luisfmaia.com/

[2] https://player.vimeo.com/video/272241245

subjects, so they could be aware of the most important traits investigated in this research. Besides, the goal behind supplying all these meticulous data about the problem was to qualify every user to evaluate the algorithms properly, even if they had never played an LBG or marginally knew the area.

The page also contained a questionnaire dedicated to assess the knowledge of users about LBGs, to investigate their perception about the transposition problem, and to verify whether they have noticed the lack of game balancing between instances of LBGs. The questionnaire is available in the Appendix A.

To conduct the evaluation, the website, along with the transposition algorithms, was hosted on the Amazon Cloud, where a *c4.2xlarge* machine using an 8 core Intel Xeon E5-2666 CPU and 15GB of memory was allocated. This hardware was selected due to the need to execute a parallel approach (Section 5.3) and to handle multiple user requests at once without loosing performance. Table 7 shows the parameters used to set the algorithms during trials. Besides, the statistical analysis was conducted using the software StatPlus version 6.7, running on a MacBook Pro (MacOS 10.13.6).

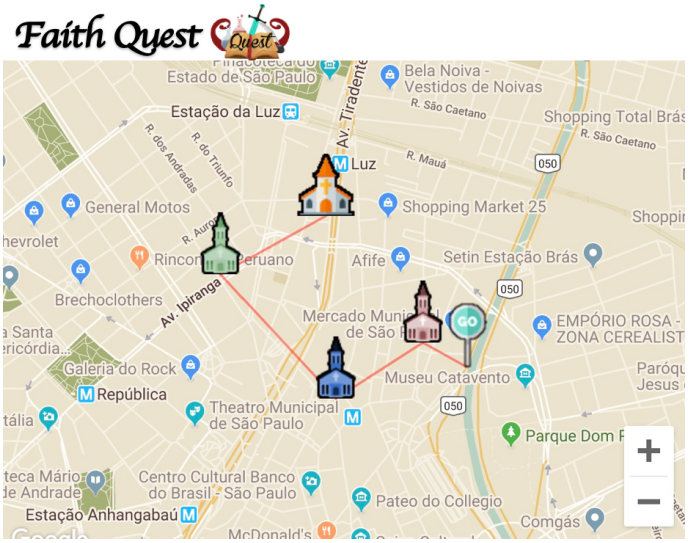
Next, the games created to compose the user evaluation are presented. To make sure the transposition works regardless of mechanics or game flow, each game was designed to be unique in its size, purpose, style, and gameplay. Before the evaluation, subjects were informed about the goals and mechanics of each game, therefore they could rightfully judge the quality of the transposed maps.

7.1.1 Faith Quest

In the game *Faith Quest*, players must follow a specific path that passes across religious places in the region the game is being played. To finish the game, it is necessary to complete the pilgrimage, so the algorithm must pick locations that are classified as religious POIs.

Figure 45 depicts an instance of the game showing a starting point and a red line connecting the religious places to indicate the route to be followed. In the original instance the following adjacency matrix was used to define the game:

Figure 45 – The game Faith Quest shows a route to follow when visiting religious sites.



Source: Author

$$A_R = \begin{vmatrix} 0 & 550 & -1 & -1 & -1 \\ -1 & 0 & 600 & -1 & -1 \\ -1 & -1 & 0 & 625 & -1 \\ -1 & -1 & -1 & -1 & 550 \\ -1 & -1 & -1 & -1 & 0 \end{vmatrix}$$

The game contains 5 (five) places to be visited, and in this particular case, the algorithm was configured to look for 25 POIs, primarily classified as religious locations, such as churches, temples, sanctuaries, etc. In the event the area chosen by the user lacks the necessary amount of religious sites, the algorithm fills the remaining spots with other places, such as schools, squares, public buildings, among others. This strategy has been successfully used in (FERREIRA *et al.*, 2019).

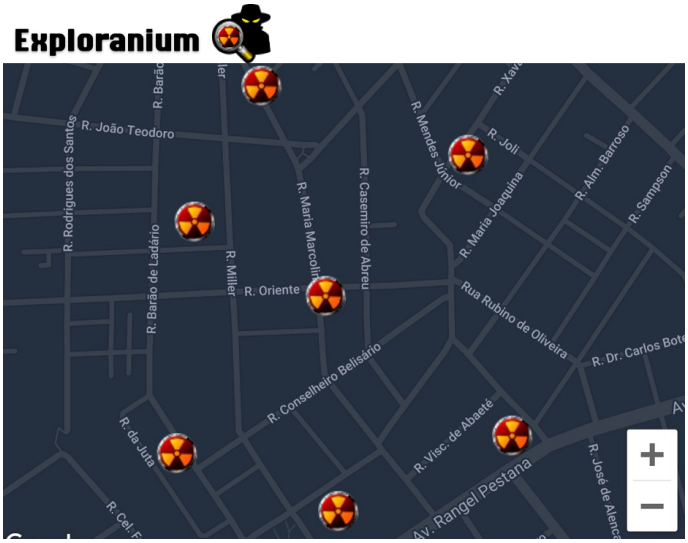
7.1.2 Exploranium

This game is set during the second world war and begins by informing that the player is a member of a secret government agency whose main focus is to hinder the development of nuclear weapons. In this case, the player's mission is to move to specific locations as fast as possible, and sabotage the equipment used in the manufacturing of those weapons.

To accomplish this task, a map containing the location of each site to be visited is handed to the player. There is no predefined sequence of places to follow, so players must

analyze their surroundings before deciding the route that allows them to visit every site in the shortest time. This type of gameplay allows users to move freely and became popular for being present in the games *Ingress* and *Pokémon GO*.

Figure 46 – Maps for the game Exploranium present locations to be visited at users' will.



Source: Author

A key characteristic of this game is the regular distribution between the 7 (seven) locations that compose the map. These locations were selected from a set of 40 candidate POIs queried in the area. In its original version, the locations are positioned evenly around a central POI (as shown in Figure 46), and its adjacency matrix was defined as:

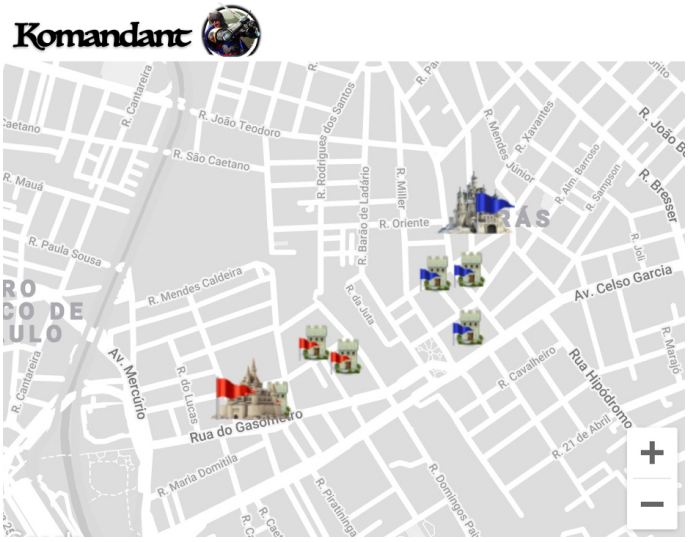
$$A_R = \begin{vmatrix} 0 & 500 & 500 & 500 & 500 & 500 & 500 \\ 500 & 0 & 500 & 850 & 1000 & 850 & 500 \\ 500 & 500 & 0 & 500 & 850 & 1000 & 850 \\ 500 & 850 & 500 & 0 & 500 & 850 & 1000 \\ 500 & 1000 & 850 & 500 & 0 & 500 & 850 \\ 500 & 850 & 1000 & 850 & 500 & 0 & 500 \\ 500 & 500 & 850 & 1000 & 850 & 500 & 0 \end{vmatrix}$$

7.1.3 Komandant

Komandant is a battle game where two players compete against each other. Each player controls a realm and their ultimate goal is to defeat the opponent. However, to win the war, it is necessary to conquer all enemy towers before attacking the castle.

For being a strategy game that makes use of the physical distribution of POIs, it is crucial for the game balancing to have the castle protected by their towers (Figure 47). Therefore, this feature must be preserved by the transposition method.

Figure 47 – In the game Komandant, the castle must be protected by their towers.



Source: Author

The game is composed of 8 (eight) POIs divided between two kingdoms, where the castles must be positioned further from each other and their respective towers must be placed in the front line of the battlefield. The transposition algorithms were configured to pick these POIs among a list of 50 candidates. Below the adjacency matrix that originates the game is shown.

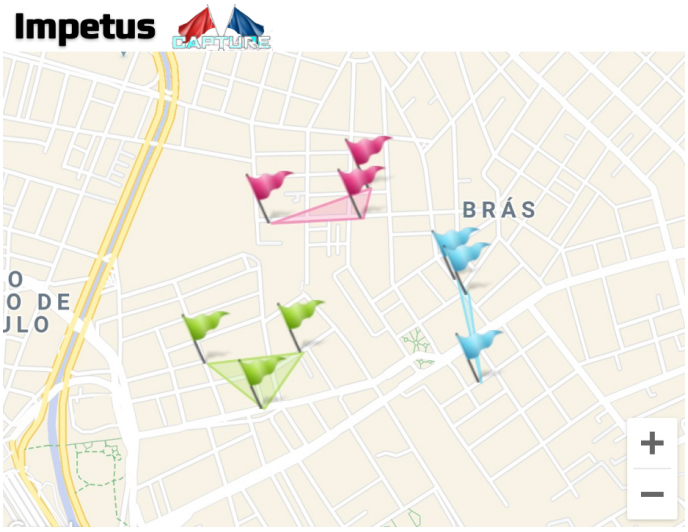
$$A_R = \begin{vmatrix} 0 & 300 & 300 & 300 & 1300 & 1000 & 1000 & 1000 \\ 300 & 0 & 300 & 300 & 1000 & 700 & 700 & 700 \\ 300 & 300 & 0 & 300 & 1000 & 700 & 700 & 700 \\ 300 & 300 & 300 & 0 & 1000 & 700 & 700 & 700 \\ 1300 & 1000 & 1000 & 1000 & 0 & 300 & 300 & 300 \\ 1000 & 700 & 700 & 700 & 300 & 0 & 300 & 300 \\ 1000 & 700 & 700 & 700 & 300 & 300 & 0 & 300 \\ 1000 & 700 & 700 & 700 & 300 & 300 & 300 & 0 \end{vmatrix}$$

7.1.4 Impetus

This game is an adaptation of the famous capture-the-flag genre, designed to be played in groups that clash for territorial dominance. The main goal of each team is to gain

territory by capturing the opponent's flags, while striving to not have their own flags captured (Figure 48).

Figure 48 – Figure showing the distribution of territory in the game Impetus



Source: Author

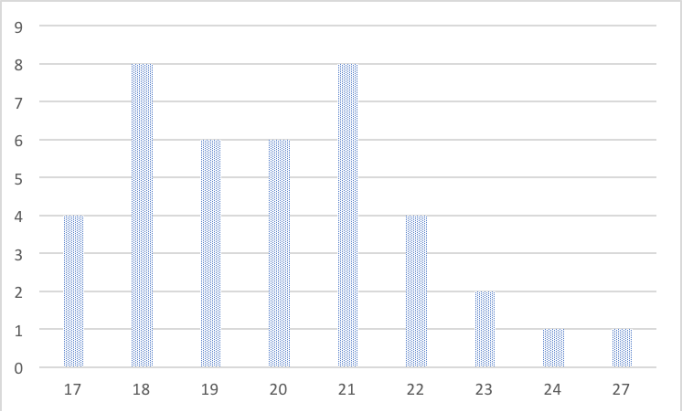
A key feature in this game is the distribution of flags to form the territory of a group, thus 9 (nine) locations must be selected out of 50 candidate POIs and divided among three opponents. Consequently, the flags belonging each team must be positioned so as to cover equivalent territorial areas. Moreover, these territories must be placed apart from each other. As a result, the following adjacency matrix was used to define the game:

$$A_R = \begin{vmatrix} 0 & 400 & 400 & 1000 & 1000 & 1000 & 1000 & 1000 & 1000 \\ 400 & 0 & 400 & 1000 & 1000 & 1000 & 1000 & 1000 & 1000 \\ 400 & 400 & 0 & 1000 & 1000 & 1000 & 1000 & 1000 & 1000 \\ 1000 & 1000 & 1000 & 0 & 400 & 400 & 1000 & 1000 & 1000 \\ 1000 & 1000 & 1000 & 400 & 0 & 400 & 1000 & 1000 & 1000 \\ 1000 & 1000 & 1000 & 400 & 400 & 0 & 1000 & 1000 & 1000 \\ 1000 & 1000 & 1000 & 1000 & 1000 & 1000 & 0 & 400 & 400 \\ 1000 & 1000 & 1000 & 1000 & 1000 & 1000 & 400 & 0 & 400 \\ 1000 & 1000 & 1000 & 1000 & 1000 & 1000 & 400 & 400 & 0 \end{vmatrix}$$

7.2 Sample

Participants were randomly recruited via e-mail from the Federal University of Ceará and the Federal Institute of Education of Maranhão, and via social networks from LBGs' communities. In total, 40 subjects participated in the trials. Their average age was 20, ranging between 17 – 27, as shown in the histogram depicted in Figure 49.

Figure 49 – Histogram showing the distribution of ages among participants.



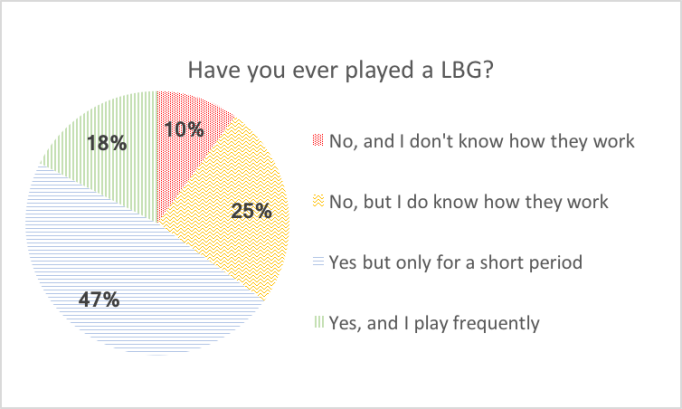
Source: Author

Users also answered a question regarding their previous experience with LBGs (Question 3 in Appendix A). In this case, 65% of participants claimed to have played LBGs, either frequently (17.5%) or for a short period (47.5%). Furthermore, 25% stated to have not played any LBG, but to know how they work, and only 10% of responses informed to be unaware of LBGs. This information is key to this evaluation, because the more participants know about these games and how they work, the easier is for them to understand the concepts and features being evaluated. Figure 50 shows the question and the percentage of each answer.

Other aspects evaluated in the questionnaire relate to the deployment of games in multiple places and the perception of fairness between games played in distinct regions. Subjects were asked if, in their opinion, current LBGs can be played everywhere, based on the LBGs they have played. The majority of users (58%) claimed to partially agree, while another 35% answered they partially or totally disagree (Figure 51).

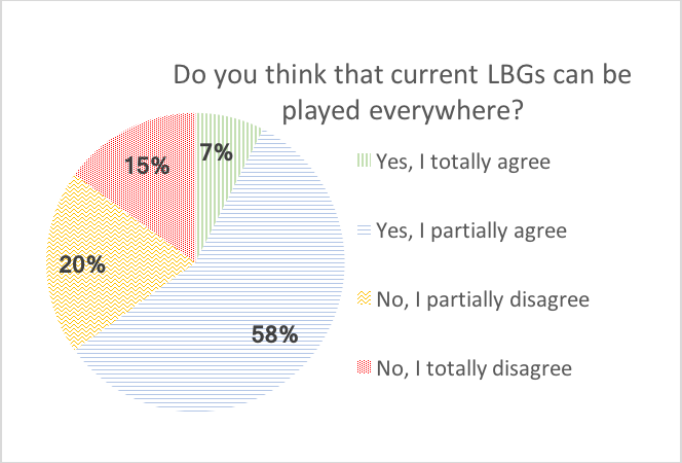
Lastly, users answered a question about the benefits players may experience in LBGs depending on their location. In this case, an overwhelming 97% of answers agree that the location has some influence to players in current LBGs. Figure 52, shows that only 3% of participants stated to totally disagree, while no one partially disagreed with this question.

Figure 50 – Chart presenting answers about previous experience with LBGs.



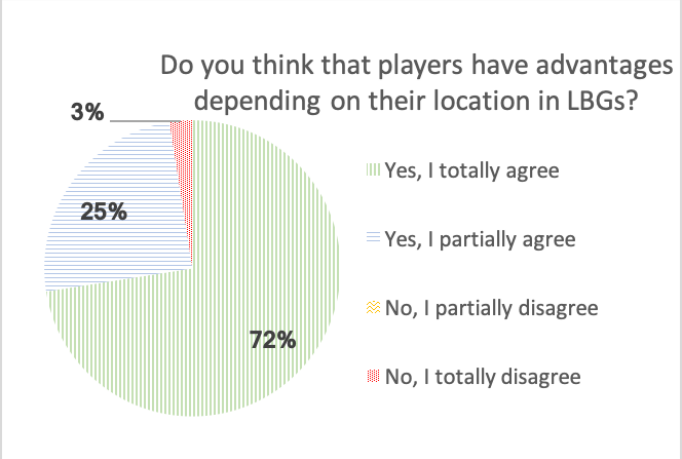
Source: Author

Figure 51 – Users answered whether they think current LBGs can be played everywhere.



Source: Author

Figure 52 – Most participants claim that location benefits players in current LBGs.



Source: Author

7.3 Procedure

After answering the questionnaire and receiving instructions about the purpose of the research, participants assessed transposed maps by comparing them with the original instance. This required users to evaluate maps of games that were generated by the approach presented in Chapter 4. In this case, for each game introduced in Section 7.1, a set of transposed maps was created using the algorithms detailed in Chapter 5, and a map generated from randomly selected sites was also added for comparison purposes. As a result, each subject assessed 16 maps (4 per game), that were displayed randomly, thus totaling 640 evaluations.

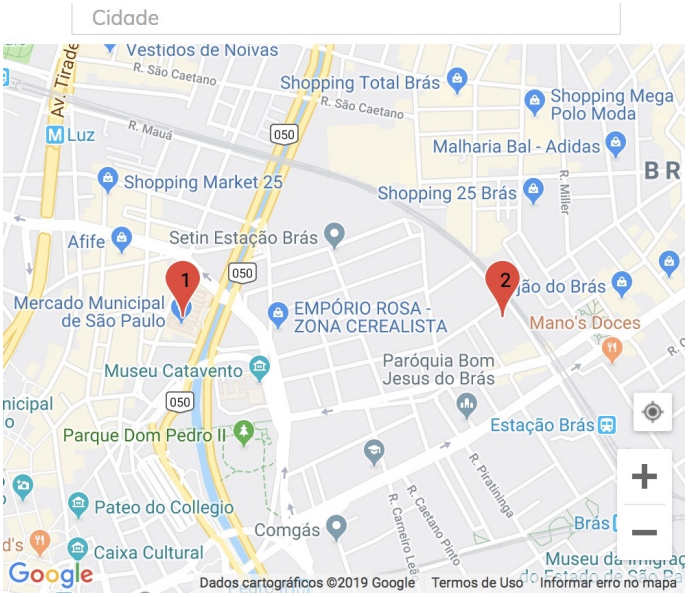
Subjects were asked to give 5 stars if they judge the transposed map has the same difficulty level of the original game, 4 stars if they think the difficulty is similar, 3 stars if the similarity is only satisfactory, 2 stars if it is bad, and 1 star if they consider the differences between maps to be terrible. Consequently, lower grades mean poor outcomes and higher scores indicate the transposition algorithm is working properly.

Besides, to provide a more reliable evaluation, the original game and the transposed maps were placed in regions the subjects are familiar with. Thus, before presenting the games, users were asked to select two well known locations apart from each other (Figure 53). The first place was used to create the original game map, and the second location served as input for the proposed method to generate the transposed game instances. In all cases, the transposition methods made use of the *Google Places API* to select candidate POIs in the new region, and queried *Google Distance Matrix API* for the distance to walk between these points.

During the evaluation, users could visualize the original map on the left and had the option to navigate through the transposed maps, randomly displayed on the right (Figure 54). For each game, a text providing context about the gameplay and a set of instructions about the evaluation was displayed below the original map. Subjects could freely navigate back and forward across the transposed maps to edit the grades they have assigned previously. In addition, the game maps were fully interactable, thus participants could drag, zoom, click and explore each map freely.

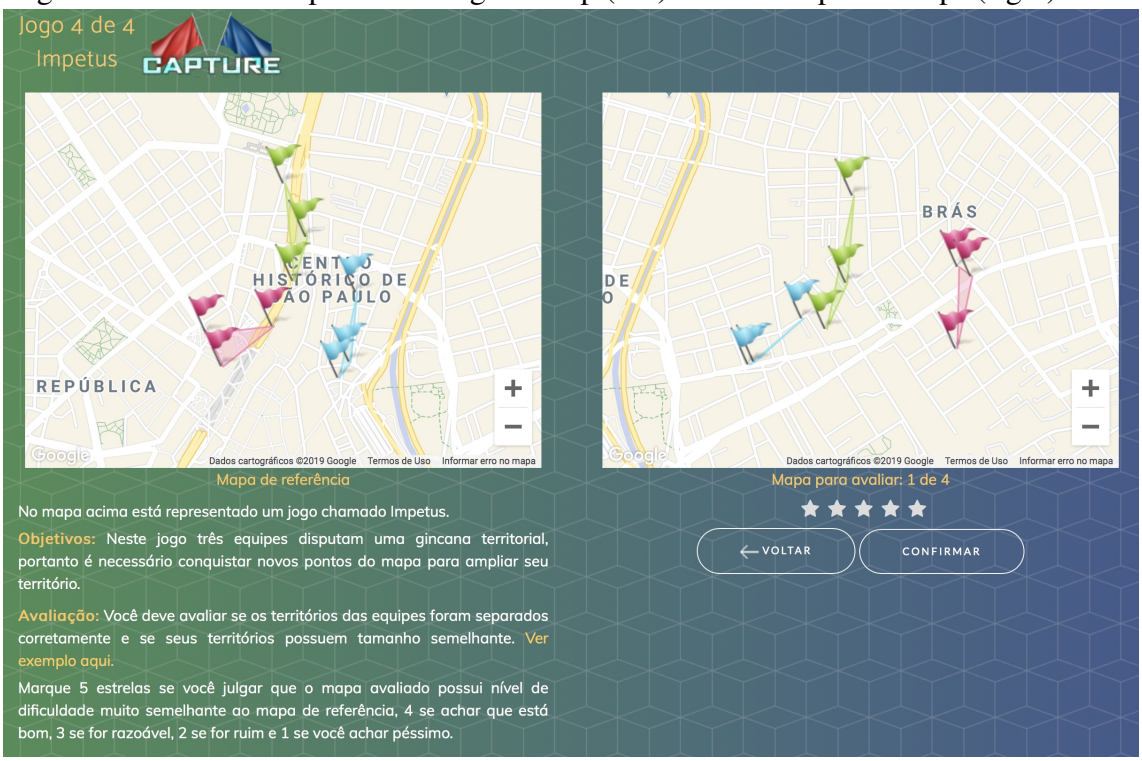
After evaluating all the games, a final page was shown thanking users for joining the experiment and asking them to report any trouble or to leave suggestions.

Figure 53 – Image depicting a user selecting distinct locations to be used in the evaluation.



Source: Author

Figure 54 – Users compared the original map (left) to the transposed maps (right).



Source: Author

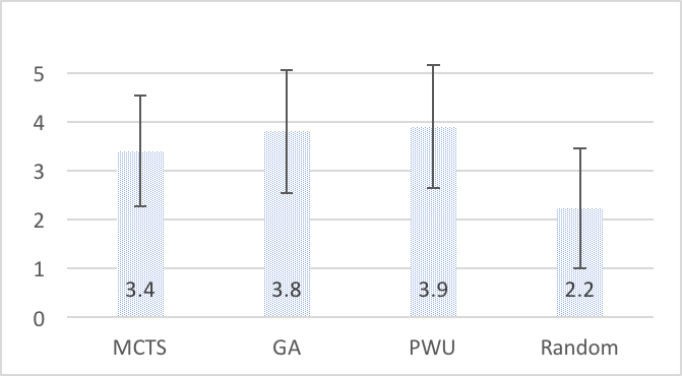
7.4 Results

In this section, the results of the user evaluation are discussed. The data collected is detailed according to each game, since they vary in size, context, and gameplay. The analysis includes the final average score assigned to maps transposed by each algorithm, their

corresponding $\mathfrak{M}$ with respect to the original game, and a comparison between $\mathfrak{I}$ and $\sigma_{\mathfrak{I}}$ of the original instances and the average values of transposed maps. In the end, a statistical analysis to determine the correlation between lower $\mathfrak{M}$ and better game maps is presented.

The first game evaluated was *Faith Quest*, and subjects were asked to assess, based on their knowledge about the terrain, whether the paths in both original and transposed maps had similar length. As a result, the PWU was the best evaluated (average score 3.9 ± 1.26), followed closely by the GA (3.8 ± 1.26). MCTS received a slightly worse grade (3.4 ± 1.13), while the random selection had a very poor evaluation (2.2 ± 1.23). Figure 55 depicts these grades in a chart, along with the error bars indicating the standard deviation.

Figure 55 – Average score for the transposed maps of the game Faith Quest.

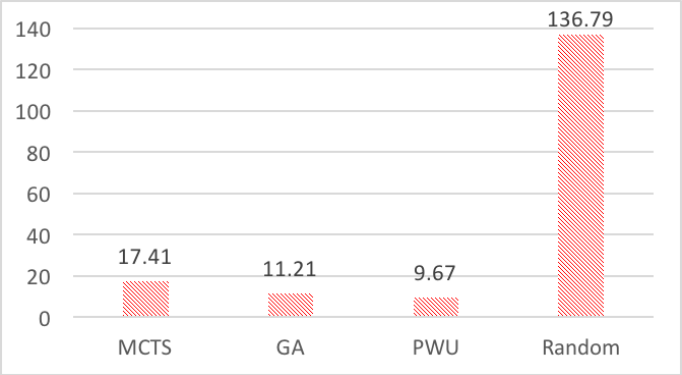


Source: Author

In addition, an analysis of the balancing difference was conducted, and results showed that the PWU, indeed, delivers better results for this particular challenge. In this case, lower values of $\mathfrak{M}$ mean the transposed map delivers similar game balancing to the original one. The GA and MCTS also delivered good results, though MCTS almost doubled $\mathfrak{M}$ when compared to PWU. Conversely, the random selection of locations generated games with poor similarity, as shown in Figure 56.

An analysis of $\mathfrak{I}$ and $\sigma_{\mathfrak{I}}$ for this game also indicates the proposed approach creates transposed maps with similar internal game balancing (Figure 57). This corroborates to the goals of the transposition algorithms, that intend to generate maps as similar as possible to the original game. In this case, the values of $\mathfrak{I}$ for the transposition algorithms are almost equal to the original instance, and the values of $\sigma_{\mathfrak{I}}$ are a little higher in the transposed maps. Thus indicating that the overall path to follow has the same length in most maps, but the edges that make up the path have a bigger distance difference. This behavior is expected, mainly due to

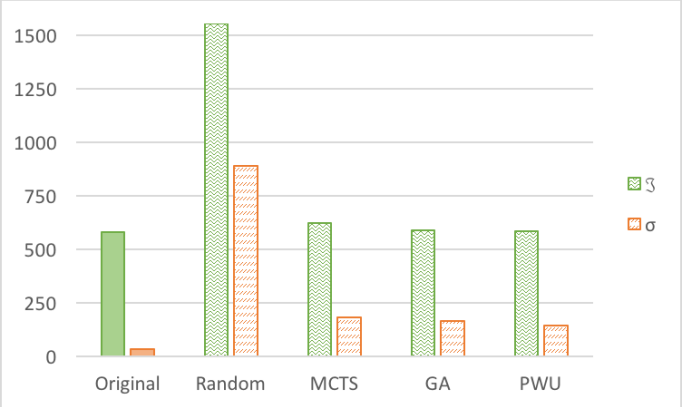
Figure 56 – Comparison between $\mathfrak{M}$ for different map transpositions of the game Faith Quest.



Source: Author

the need to select religious locations to form the new game map, hence the method favors the selection of places by their type even when their location is not ideal.

Figure 57 – Chart depicting $\mathfrak{I}$ and $\sigma_{\mathfrak{I}}$ for the original instance of the game Faith Quest and the average value of the transposed maps.



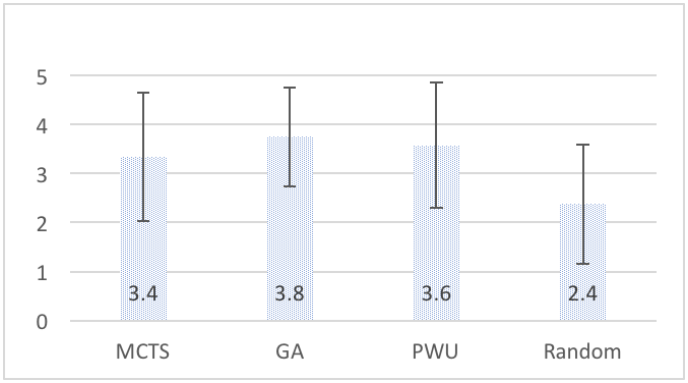
Source: Author

Next, participants evaluated the transposed maps for the game *Exploranium*. Subjects were asked to judge whether the marked sites were distributed evenly from one another. The maps contained seven locations each, and the original game was used as reference during comparisons. In this case, algorithms operated more closely, with the GA scoring slightly better (3.8 ± 1.0) than PWU (3.6 ± 1.28) and MCTS (3.4 ± 1.31). The random approach again achieved lower grades (2.4 ± 1.21) as depicted in Figure 58.

Similar to the results obtained by *Faith Quest*, Figure 59 reinforces the existing correlation between smaller $\mathfrak{M}$ and better evaluated maps. However, in the game *Exploranium* it is clear that the proposed algorithms delivered more equivalent results.

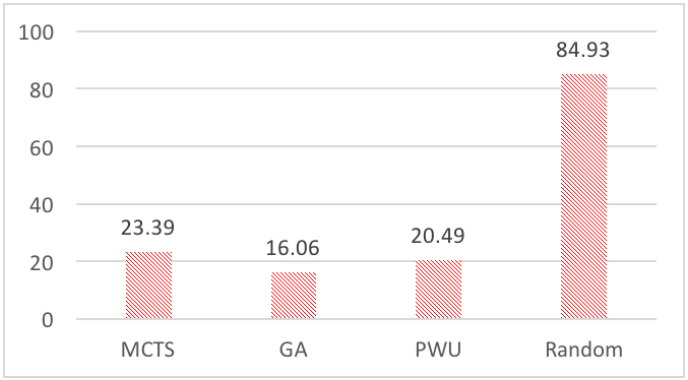
In the case of $\mathfrak{I}$ and $\sigma_{\mathfrak{I}}$, all the transposition algorithms delivered solutions with

Figure 58 – Average score for the transposed maps of the game Exploranium.



Source: Author

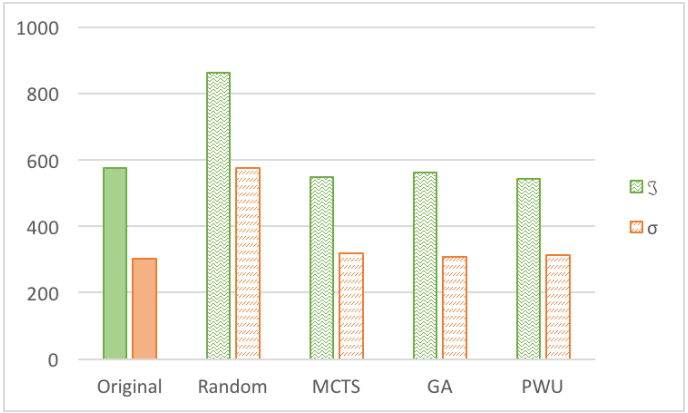
Figure 59 – Comparison between $\mathfrak{M}$ for different map transpositions of the game Exploranium.



Source: Author

internal game balancing very similar to the original game, as shown in Figure 60. Conversely, the random approach produced game maps with a considerable difference as its $\sigma_{\mathfrak{I}}$ is nearly the double of the original game, thus contributing to its lower user rating.

Figure 60 – Chart depicting $\mathfrak{I}$ and $\sigma_{\mathfrak{I}}$ for the original instance of the game Exploranium and the average value of the transposed maps.

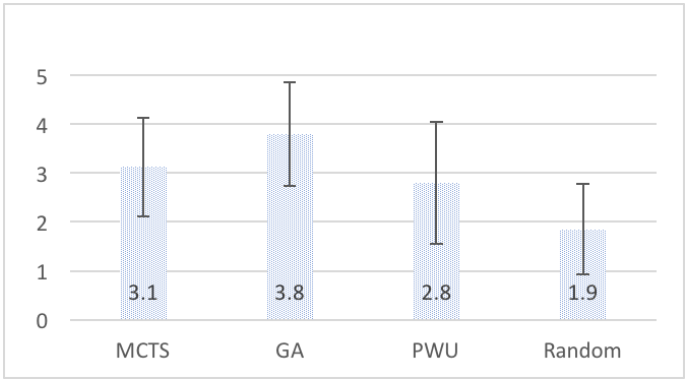


Source: Author

The third game evaluated was *Komandant*, and due to its inherent dependency on the positioning of resources, the transposition algorithms must preserve the disposition of castles and towers as best as possible. Consequently, subjects were asked to assess if towers and castles were positioned similarly to a reference game instance.

The data collected indicates the GA worked significantly better than other methods, thus presenting minor $\mathfrak{M}$ and higher score (3.8 ± 1.06). In this scenario, MCTS (3.1 ± 1.01) surpassed PWU ($2.81.24\pm$), while the random selection continued to deliver the worst results (1.9 ± 0.92). Figure 61 illustrates this trend.

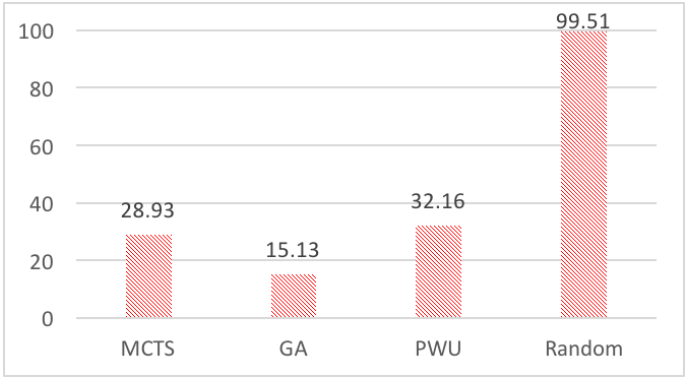
Figure 61 – Average score for the transposed maps of the game Komandant.



Source: Author

Besides, the comparison between the $\mathfrak{M}$ corroborates to the scores achieved by each algorithm, as the GA presented the lower rate, followed by MCTS, PWU and the random approach (Figure 62).

Figure 62 – Comparison between $\mathfrak{M}$ for different map transpositions of the game Komnadant.

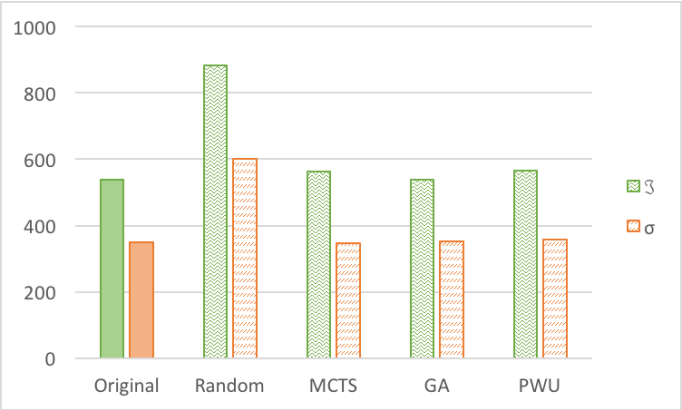


Source: Author

Likewise the previous game, the transposition algorithms generated maps with $\mathfrak{I}$ and $\sigma_{\mathfrak{I}}$ are very similar to the original game, while the random approach produced maps with big

differences in internal game balancing. Figure 63 illustrates this trend.

Figure 63 – Chart depicting $\mathfrak{I}$ and $\sigma_{\mathfrak{I}}$ for the original instance of the game Komandant and the average value of the transposed maps.

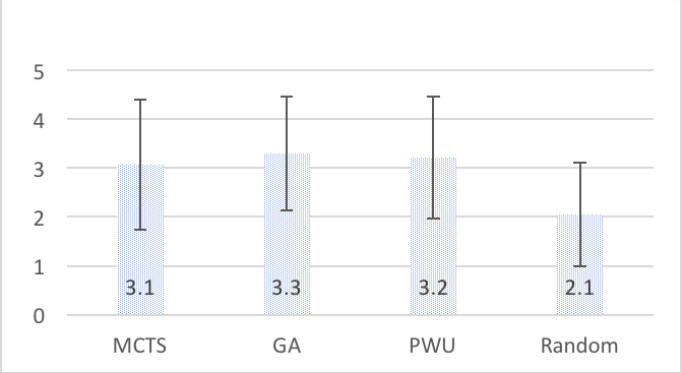


Source: Author

The last game evaluated was *Impetus*, which is an LBG adaptation of capture-the-flag games that supports multiple players competing in groups. Therefore, participants were asked to evaluate if the flags were distributed properly, so as to create territories with similar size and apart from each other.

Results showed that the three proposed algorithms delivered a similar, albeit ordinary performance, where GA received better grades (3.3 ± 1.16), followed by PWU (3.2 ± 1.25) and *MCTS* (3.1 ± 1.33). Once more, the random selection had the lowest rates in the comparison (2.1 ± 1.06), as shown in Figure 64.

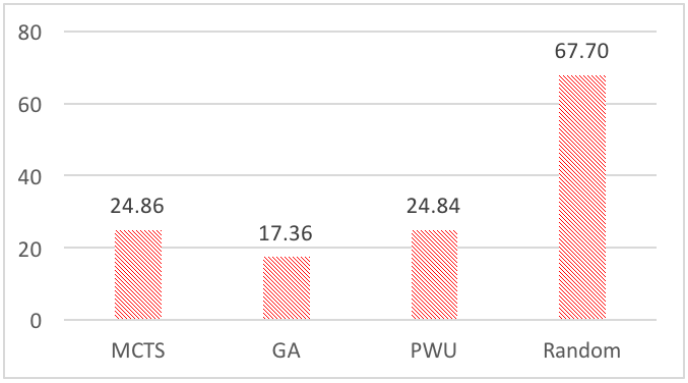
Figure 64 – Average score for the transposed maps of the game Impetus.



Source: Author

Again, the graph depicting the comparison between $\mathfrak{M}$ for the methods corresponds to the rates provided by users, as depicted in Figure 65.

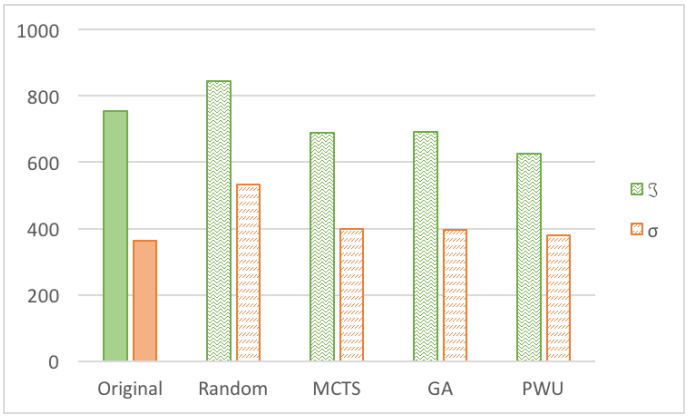
Figure 65 – Comparison between $\mathfrak{M}$ for different map transpositions of the game Impetus.



Source: Author

Regarding internal game balancing, the transposition algorithms have produced games with slightly smaller $\mathfrak{I}$ but with very similar $\sigma_{\mathfrak{I}}$. This indicates that the distances to move between POIs have similar variation, however, in this game, the overall distance to move between all flags is shorter than in the original map.

Figure 66 – Chart depicting $\mathfrak{I}$ and $\sigma_{\mathfrak{I}}$ for the original instance of the game Impetus and the average value of the transposed maps.



Source: Author

As highlighted by the results previously shown, the transposition algorithms were better evaluated than the random selection of points in all cases. However, to make sure there is a statistically significant difference between the methods, lets consider the following null hypothesis:

H_0: The method has no significant effect on the score of transposed maps.

Accordingly, since the evaluation consisted in four distinct methods (MCTS,PWU, GA, and random selection), a One-way ANOVA was run with the rates provided by users for

each game. This analysis simultaneously compares the rating of all the methods to indicate whether there is a significant difference between the transposition algorithms. Table 17 shows that hypothesis H_0 must be rejected in all cases, as $F > F_{critical}(2.66)$ and $p - value < 0.05$ for each game tested.

Table 17 – Table presenting the results of One-way ANOVA comparing MCTS, PWU, GA, with random selection.

	Faith Quest	*Exploranium*	*Komandant*	*Impetus*
$p - value$	4.82×10^{-9}	3.07×10^{-6}	2.16×10^{-12}	9.89×10^{-6}
F	15.8451	10.3258	23.01935	9.36828

Source: Author

The statistical analysis showed that only in the game *Komandant* there was a significant difference ($F < F_{critical}(3.07)$), as subjects preferred maps transposed by the GA (Table 18).

Table 18 – Table presenting the results of One-way ANOVA comparing MCTS, PWU, with GA.

	Faith Quest	*Exploranium*	*Komandant*	*Impetus*
$p - value$	0.15609	0.33481	0.00039	0.71444
F	1.8871	1.10448	8.38832	0.33722

Source: Author

7.5 Discussion

The previous section presented the results obtained from the user evaluation. The results clearly indicate that users preferred maps transposed using the proposed algorithms. Moreover, the statistic analysis showed that there was, indeed, a significant difference between the evaluation of maps transposed using the proposed algorithms and the random selection of sites. This information allows us to link higher scores to smaller values of $\mathfrak{M}$, thus indicating that the metric is functional to assess the game balancing between LBGs.

Besides, in most cases, the transposed maps received scores higher than 3 (three), except for the PWU algorithm in the *Komandant* trial. In this particular case, the PWU pruned large branches of the search tree while searching for the best matches, thus it ended up with sub-optimal solutions. By comparison, the average time PWU took to compute solutions was 0.59 seconds, whereas MCTS and GA used 4.93 and 5.33 seconds, respectively. A simple way to improve the quality of results of PWU is to reduce the rate (δ) to which the threshold (τ) is reduced, thus the method becomes more accurate at the expense of increasing its processing time, as shown in Section 5.3.

The fact that the games designed for these trials had varying sizes and gameplay indicate that the method is flexible, and can potentially be used in many types of LBGs. Furthermore, the average time spent for the methods to calculate the transposition did not exceed 5.5 seconds, hence allowing the algorithms to be used on demand.

During the tests, 46 distinct locations were selected by users, culminating in the transposition of 800 game maps (including the reference maps) and more than 3.000 POIs being analyzed by the algorithms. Among all these numbers, there has not been a single case where the method failed to generate the transposed version of a map. These data reinforces that the method elaborated in this work can potentially be used to automatically execute the balanced transposition of maps of LBGs.

7.6 Threats to Validity

The main concern about this evaluation relates to the quality of the answers. It was necessary to ensure participants assessed the maps correctly, according to the instructions. Thus, many content was provided (texts, videos, and images) to explain the purpose of the evaluation and to highlight the traits being analyzed. However, it is still possible that subjects may have skipped the introductory parts or misunderstood some aspects of the evaluation. After concluding the evaluation, in the last page, half of subjects left comments, however, only 03 (three) claimed they were confused about how to assess each game map.

Another threat to the trials is the reliance on data gathered from external APIs (in this case *Google Places* and *Google Distance Matrix*), since the locations selected to build the game and the game balancing in both original and transposed instances is estimated using this information. Consequently, if the data queried from external sites is not accurate, nor will be the estimated game balancing and its evaluation.

The tests used games with different features, however, all games assessed contained less than 10 POIs. Unquestionably, it would be ideal to evaluate games with more POIs, albeit the complexity of these games would also make it harder for participants to evaluate bigger maps.

Finally, the game balancing when playing the games can be distinct from the one perceived by users when assessing the maps. Consequently, an ideal evaluation would request users to play each map so they would render their opinion according to the real experience.

7.7 Conclusion

This chapter detailed the evaluation used to validate the balanced transposition of maps for LBGs. The trials were conducted with 40 subjects using an online website, and four LBGs were designed specifically to be used in this evaluation.

The method and games were introduced, and the procedure adopted to apply the tests was detailed. In resume, users compared the original game map to game maps generated by distinct transposition algorithms. Next, the results for each game were presented and discussed individually. Results, evidenced through a statistical analysis, the preference for maps transposed using the proposed algorithms over maps generated randomly.

In addition, the data generated during tests indicate that $\mathfrak{M}$ is directly linked to the better evaluation of maps. Therefore, this evaluation endorses the fact that the proposed approach can be used with different types of LBGs, regardless of their size and gameplay.

8 CONCLUSIONS

This thesis addressed the challenge of transposing maps of LBGs while focusing on maintaining the original game balancing. To achieve this goal, a game model was developed to encode LBGs regardless of their style, gameplay and number of POIs, novel metrics to gauge game balancing in these games were introduced, and algorithms to tackle the transposition as a WGMP were implemented.

This chapter closes the thesis and is organized as follows. Section 8.1 provides an overview of this research, and Section 8.2 presents the main contributions of this thesis. Next, Section 8.3 revisits the research hypothesis, and Section 8.4 discusses the limitations of this work. Finally, Section 8.5 points to works to be developed in the future.

8.1 Overview

There are many motivations behind this work, such as the rising popularity of LBGs and the benefits they bring to health and entertainment, the challenge of designing better, cheaper and fairer games, and the possibility to increase the reach of LBGs to as many places as possible. Therefore, the challenge of creating a PCG method to conduct the balanced transposition of maps of LBGs was tackled.

The first step in this work consisted in devising measurements to be used by game designers and developers for assessing LBGs. Therefore, two metrics were proposed, the *Internal Difficulty Level* ($\mathfrak{I}$) and the *Minimum Balance Difference* ($\mathfrak{M}$). The former is indicated when creating new games as it gauges differences in game balancing within the game, and the latter is used to compare the game balancing between two instances of the same game in distinct places, i.e. their maps.

Next, the approach in charge of executing the transposition was modelled. It consists in creating a generic game model from the map to be transposed and collecting data about POIs in the area where the transposed map must be placed, then an algorithm selects the best locations to form a new game model as similar to the original as possible, so that the new game model encoding the transposed locations can be converted back to the game, thus generating the transposed map of the LBG.

Three algorithms were implemented to select the POIs that will compose the games, a MCTS method previously used in (MAIA *et al.*, 2017) and (FERREIRA *et al.*, 2017), a novel

approach called *Parallel Weighted Ullmann* that is based on the Ullmann' algorithm, and a Genetic Algorithm.

These algorithms were evaluated and an analysis of their speed and quality when submitted to varying types of inputs was presented. Hence, the evaluation discusses the strengths and weaknesses of each algorithm and highlights the best applications for each approach.

An evaluation was also conducted with users to validate the quality of maps transposed using the proposed method. As a result, it was clear that users preferred maps generated by this work, as opposed to a random selection of locations. Furthermore, the data generated from the trials establishes a clear relation between the difference in the game balancing measured by $\mathfrak{M}$ and the rating provided by users.

In a nutshell, this work has the potential to enhance the development of LBGs by reducing the cost and complexity associated with it. Moreover, the proposed approach opens the possibility for these games to deliver new modes of competition. Since the possibility to create maps with similar game balancing enables players to compete or interact within the same virtual environment regardless of their location.

Finally, there are social aspects linked to the adoption of this method in the development of LBGs as most of these games do not include POIs in impoverished neighborhoods and rural areas. Hence, this work also tackles this issue by shifting the focus from playing where the game is deployed to deploying where the players are.

8.2 Contributions

The main contributions of this work are summarized below:

- **Internal Difficulty Level** ($\mathfrak{I}$). A measurement to assess variations in game balancing within a game;
- **Minimum Balance Difference** ($\mathfrak{M}$). A measurement to gauge the difference in balancing between different instances of a game;
- The design of a PCG approach that can handle multiple types of LBGs and conduct a transposition of their maps, while striving to keep the game balancing;
- **Parallel Weighted Ullmann**. A novel algorithm to tackle the GMP for weighted graphs that is based on the Ullmann's algorithm.

Besides, this research has been involved in the publication of 4 (four) papers in conferences and journals. The game model used to represent the games was originated from the

models presented in (SILVA *et al.*, 2017) and (FERREIRA *et al.*, 2017). The game balancing measurements ($\mathfrak{I}$ and $\mathfrak{M}$) were first introduced in (MAIA *et al.*, 2017). Moreover, the MCTS algorithm was used to transpose the map of a game in (FERREIRA *et al.*, 2019) and to improve game balancing in the game *Pokémon GO* (MAIA *et al.*, 2017).

Additionally, another 11 papers were published along the development of this work, mostly in the field of Games and Game Development, but some related to Augmented Reality and Software Engineering.

8.3 Revisiting the Research Hypothesis and Research Questions

The research hypothesis that guided this thesis was presented in Chapter 1. It claims that "It is possible to create a PCG method that transposes maps of LBGs while preserving their game balancing."

Based on the results collected from both the empirical evaluation (Chapter 6) and the user evaluation (Chapter 7), one can conclude the hypothesis was **accepted**, since the method comprised of measurements, algorithms and model were tested in multiple locations and with LBGs presenting varying styles. In a nutshell, this thesis managed to devise a PCG approach capable of conducting an automatic transposition of maps that focus on preserving the game balancing of the original instance.

Regarding the research questions, the first one (RQ1) required the development of the two measurements ($\mathfrak{I}$ and $\mathfrak{M}$) to estimate and compare the game balancing in LBGs. The second question (RQ2) concerned the development of a PCG method to automatically transpose maps of LBGs (Chapter 4) while enforcing the same balancing level among all instances, as presented by the problem formulation and algorithms in Chapter 5. Lastly, the answer to the last research question (RQ3) was obtained by the successful validation of the proposed approach with many users assessing the transposition of distinct types of LBGs to multiple locations.

8.4 Limitations

This work has shown promising results to address the challenge of transposing maps of LBGs. However, it relies on data provided by third party services, that can provide unreliable or outdated information. Consequently, the correctness of POIs that compose the transposed map and the accuracy of the information used to estimate the game balancing are fundamental to the

success of the method. Thus, this dependency on external entities must be faced as a limitation.

Furthermore, although current location APIs are composed of a massive database of POIs spread throughout the globe, there are still remote areas that have few or no points registered. Thus, it is not possible to gather places and data necessary to operate the transposition.

Another restriction in this approach is the resistance to updates in the transposed map. In this case, if somehow the cost to move between a few POIs is altered and the game must change to adjust game balancing, the method does not allow the game to be partially updated. Consequently, any update to a map requires the creation of an entire new instance.

Additionally, the current game model encodes only one information in the graphs, thus if more than one feature influences the game balancing, they must be combined into a single weight. This is particularly restrictive if a game relies on multiple aspects to determine game balancing. This limitation is also discussed in future works.

8.5 Future Work

There are features that can be improved in this work, and can evidently be developed in the future. For instance, the PWU algorithm can be improved to include a non-deterministic step that selects branches in large search trees to enable the method to process large graphs. Besides, the GA can have some of its steps implemented in parallel, so the performance of the algorithm can be enhanced.

Another aspect that can be improved in the future is the possibility of adapting the method to update parts of maps in real-time. In this case, it is necessary to modify the game model and the transposition algorithm to signal parts of the map that must be altered. Hence, it will be possible to skip POIs that must be preserved and focus on the ones that must be updated.

It is also possible to adapt this approach to include graph consensus, so it will be possible to use multiple features when determining game balancing. As a result, the transposition could benefit from this trait to create maps that can be balanced according to multiple aspects, such as distance, time, calories, etc.

Finally, there are ongoing activities being developed as part of this work. First, a paper presenting the PWU as a fast approach for finding near optimal solutions to the WGMP, and another article detailing the transposition method and the evaluations conducted in this work. Furthermore, a game incorporating the features and technologies originated from this work is being designed.

REFERENCES

ABUZURAIQ, A. M. On using graph partitioning with isomorphism constraint in procedural content generation. In: **Proceedings of the 12th International Conference on the Foundations of Digital Games**. New York, NY, USA: ACM, 2017. (FDG '17), p. 77:1–77:10. ISBN 978-1-4503-5319-9.

ADRIAN, D. F. H.; COSÍO, A. L. S. G. An approach to level design using procedural content generation and difficulty curves. In: **2013 IEEE Conference on Computational Inteligence in Games (CIG)**. Niagara Falls, ON, CA: IEEE Computer Society, 2013. p. 1–8. ISSN 2325-4289.

AKSELSEN, A. S.; KRISTIANSEN, K. **Pervasive games in modern mobile technology: A user study**. Master's Thesis — Institutt for datateknikk og informasjonsvitenskap, Trondheim, Norway, 2010. Disponível em: http://hdl.handle.net/11250/252204. Acesso em: 20 dez. 2018.

AKUTSU, T.; NAGAMOCHI, H. Comparison and enumeration of chemical graphs. **Computational and Structural Biotechnology Journal**, Elsevier, *[s.l.]*, v. 5, n. 6, p. e201302004, 2013. ISSN 2001-0370.

ALAVESA, P.; OJALA, T. Street art gangs: location based hybrid reality game. In: ACM. **Proceedings of the 14th International Conference on Mobile and Ubiquitous Multimedia**. Linz, Austria: ACM, 2015. p. 64–74.

ALHA, K.; KOSKINEN, E.; PAAVILAINEN, J.; HAMARI, J. Why do people play location-based augmented reality games: A study on pokémon go. **Computers in Human Behavior**, Elsevier, *[s.l.]*, v. 93, p. 114 – 122, 2019. ISSN 0747-5632.

ALMOHAMAD, H. A.; DUFFUAA, S. O. A linear programming approach for the weighted graph matching problem. **IEEE Trans. Pattern Anal. Mach. Intell.**, IEEE Computer Society, Washington, DC, USA, v. 15, n. 5, p. 522–525, May 1993. ISSN 0162-8828.

AMIN, M. S.; BHATTACHARJEE, A.; JAMIL, H. A cytoscape based framework for efficient sub-graph isomorphic protein-protein interaction motif lookup. In: **Proceedings of the 2010 ACM Symposium on Applied Computing**. New York, NY, USA: ACM, 2010. (SAC '10), p. 1572–1576. ISBN 978-1-60558-639-7.

ANDRADE, G.; RAMALHO, G.; GOMES, A. S.; CORRUBLE, V. Dynamic game balancing: An evaluation of user satisfaction. In: LAIRD, J. E.; SCHAEFFER, J. (Ed.). **AIIDE - Conference on Artificial Intelligence and Interactive Digital Entertainment**. Marina del Rey, CA, USA: The AAAI Press, 2006. p. 3–8. ISBN 978-1-57735-235-8.

ANDRADE, G.; RAMALHO, G.; SANTANA, H.; CORRUBLE, V. Automatic computer game balancing: A reinforcement learning approach. In: **Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems**. New York, NY, USA: ACM, 2005. (AAMAS '05), p. 1111–1112. ISBN 1-59593-093-0.

AREA, G. **Shadow Cities Wiki | Fandom powered by Wikia**. 2016. Disponível em: http://shadowcities.wikia.com/wiki/Shadow_Cities_Wiki. Acesso em: 12 nov. 2016.

ARISGAMES. **ARIS - Mobile Learning Experiences - Creating educational games on the iPhone**. 2014. Disponível em: https://arisgames.org. Acesso em: 20 nov. 2016.

ARKENSON, C.; CHOU, Y.-Y.; HUANG, C.-Y.; LEE, Y.-C. Tag and seek: a location-based game in tainan city. In: ACM. **Proceedings of the first ACM SIGCHI annual symposium on Computer-human interaction in play**. Toronto, ON, CA: ACM, 2014. p. 315–318.

ASHLOCK, D. Automatic generation of game elements via evolution. In: **Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games**. Copenhagen, Denmark: IEEE Computer Society, 2010. p. 289–296. ISSN 2325-4270.

AUWATANAMONGKOL, S. Inexact graph matching using a genetic algorithm for image recognition. **Pattern Recogn. Lett.**, Elsevier Science Inc., New York, NY, USA, v. 28, n. 12, p. 1428–1437, Sep. 2007. ISSN 0167-8655.

BATCHELOR, J. **Next Games laying off 26 staff to cut costs**. 2019. Disponível em: https://www.gamesindustry.biz/articles/2019-02-15-next-games-laying-off-26-staff-to-cut-costs. Acesso em: 20 fev. 2019.

BENGOETXEA, E. **Inexact Graph Matching Using Estimation of Distribution Algorithms**. Phd Thesis (PhD Thesis) — Ecole Nationale Supérieure des Télécommunications, Paris, France, Dec 2002.

BHATTACHARJEE, A.; JAMIL, H. M. Wsm: A novel algorithm for subgraph matching in large weighted graphs. **J. Intell. Inf. Syst.**, Kluwer Academic Publishers, Hingham, MA, USA, v. 38, n. 3, p. 767–784, Jun. 2012. ISSN 0925-9902.

BLANKSTEIN, A.; GOLDSTEIN, M. Parallel subgraph isomorphism. **MIT Computer Science and Artificial Intelligence Laboratory**, Cambridge, MA, USA, 2010.

BOWSER, A. E.; HANSEN, D. L.; RAPHAEL, J.; REID, M.; GAMETT, R. J.; HE, Y. R.; ROTMAN, D.; PREECE, J. J. Prototyping in place: a scalable approach to developing location-based apps and games. In: ACM. **Proceedings of the SIGCHI Conference on Human Factors in Computing Systems**. Paris, France: ACM, 2013. p. 1519–1528.

BROWNE, C. B.; POWLEY, E.; WHITEHOUSE, D.; LUCAS, S. M.; COWLING, P. I.; ROHLFSHAGEN, P.; TAVENER, S.; PEREZ, D.; SAMOTHRAKIS, S.; COLTON, S. A survey of monte carlo tree search methods. **IEEE Transactions on Computational Intelligence and AI in Games**, *[s.l.]*, v. 4, n. 1, p. 1–43, March 2012. ISSN 1943-068X.

CARMOSINO, I.; BELLOTTI, F.; BERTA, R.; GLORIA, A. D.; SECCO, N. A game engine plug-in for efficient development of investigation mechanics in serious games. **Entertainment Computing**, Elsevier, *[s.l.]*, v. 19, p. 1–11, 2017.

CARRIGY, T.; NALIUKA, K.; PATERSON, N.; HAAHR, M. Design and evaluation of player experience of a location-based mobile game. In: ACM. **Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries**. Reykjavik, Iceland: ACM, 2010. p. 92–101.

CHANG, K. P.; HUANG, Y. W.; HSUEH, S. Y.; CHEN, Y. T.; HUANG, S. N.; CHEN, C.-H.; CHIEN, S.-F. Hidden lion: a location based app game of sword lion searching. In: ACM. **Proceedings of the first ACM SIGCHI annual symposium on Computer-human interaction in play**. Toronto, ON, CA: ACM, 2014. p. 323–326.

CHATZIGIANNAKIS, I.; MYLONAS, G.; AKRIBOPOULOS, O.; LOGARAS, M.; KOKKINOS, P.; SPIRAKIS, P. The"hot potato"case: challenges in multiplayer pervasive games based on ad hoc mobile sensor networks and the experimental evaluation of a prototype game. **arXiv preprint arXiv:1002.1099**, *[s.l.]*, 2010.

CHOI, J.; YOON, Y.; MOON, B.-R. An efficient genetic algorithm for subgraph isomorphism. In: **Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation**. New York, NY, USA: ACM, 2012. (GECCO '12), p. 361–368. ISBN 978-1-4503-1177-9.

CICIRELLO, V. A. **Survey of Graph Matching Algorithms**. Philadelphia, PA, 1999. Disponível em: https://www.cicirello.org/publications/survey-1999.pdf. Acesso em: 20 jan. 2016.

COMPTON, K.; MATEAS, M. Procedural level design for platform games. In: **Proceedings of the Second AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment**. Marina del Rey, CA, USA: AAAI Press, 2006. (AIIDE'06), p. 109–111.

COOK, D. J.; HOLDER, L. B. **Mining Graph Data**. USA: John Wiley & Sons, Inc., 2006. ISBN 0471731900.

CORTES, J.; MARTíNEZ, S.; BULLO, F. Robust rendezvous for mobile autonomous agents via proximity graphs in arbitrary dimensions. **IEEE Transactions on Automatic Control**, *[s.l.]*, v. 51, n. 8, p. 1289–1298, Aug 2006. ISSN 0018-9286.

COULOM, R. Efficient selectivity and backup operators in monte-carlo tree search. In: **Proceedings of the 5th International Conference on Computers and Games**. Berlin, Heidelberg: Springer-Verlag, 2007. (CG'06), p. 72–83. ISBN 3-540-75537-3, 978-3-540-75537-1.

CSIKSZENTMIHALYI, M. **Flow: The Psychology of Optimal Experience**. NY, NY, USA: New York: Harper & Row, 1990.

CSIKSZENTMIHALYI, M. **Beyond Boredom and Anxiety: Experiencing Flow in Work and Play**. *[s.l.]*: Wiley, 2000. ISBN 9780787951405.

DORMANS, J. Adventures in level design: Generating missions and spaces for action adventure games. In: **Proceedings of the 2010 Workshop on Procedural Content Generation in Games**. New York, NY, USA: ACM, 2010. (PCGames '10), p. 1:1–1:8. ISBN 978-1-4503-0023-0.

FALSTEIN, N. The flow channel. **Game Developer Magazine**, *[s.l.]*, v. 11, n. 05, 2004.

FERREIRA, C.; MAIA, L. F.; SALLES, C.; TRINTA, F.; VIANA, W. A model-based approach for designing location-based games. In: **2017 16th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)**. Curitiba, PR, BR: *[s.n.]*, 2017. p. 29–38. ISSN 2159-6662.

FERREIRA, C.; MAIA, L. F.; SALLES, C. de; TRINTA, F.; VIANA, W. Modelling and transposition of location-based games. **Entertainment Computing**, *[s.l.]*, v. 30, p. 100295, 2019. ISSN 1875-9521.

FERREIRA, C.; SALLES, C.; SANTOS, L.; TRINTA, F.; VIANA, W. Towards a model and a textual representation for location-based games. In: ACM. **Proceedings of the 2017 ACM Symposium on Document Engineering**. Valletta, Malta: ACM, 2017. p. 97–100.

FLINTHAM, M.; GREENHALGH, C.; LODGE, T.; CHAMBERLAIN, A.; PAXTON, M.; JACOBS, R.; WATKINS, M.; SHACKFORD, R. A case study of exploding places, a mobile location-based game. In: ACM. **Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology**. Lisbon, Portugal: ACM, 2011. p. 30.

FOURTHIRTYTHREE. **Ghostbusters World**. 2018. Disponível em: http://www.433.co.kr. Acesso em: 20 dez. 2018.

GAMES4ALL. **SpecTrek**. 2016. Disponível em: http://www.spectrekking.com. Acesso em: 20 dez. 2018.

GLASS, K. R.; MORKEL, C.; BANGAY, S. D. Duplicating road patterns in south african informal settlements using procedural techniques. In: **Proceedings of the 4th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa**. New York, NY, USA: ACM, 2006. (AFRIGRAPH '06), p. 161–169. ISBN 1-59593-288-7.

GUARDIA, D. L.; ARRIGO, M.; GIUSEPPE, O. D. A location-based serious game to learn about the culture. **Proceedings of the 2nd International Conference The Future of Education**, Florence, Italy, 2012.

GUO, B.; FUJIMURA, R.; ZHANG, D.; IMAI, M. Design-in-play: improving the variability of indoor pervasive games. **Multimedia Tools and Applications**, Springer, *[s.l.]*, v. 59, n. 1, p. 259–277, 2012.

GUO, H. **Concepts and Modelling Techniques for Pervasive and Social Games**. Phd Thesis (PhD Thesis) — Norwegian University of Science and Technology, Trondheim, Norway, 2015. Disponível em: https://pdfs.semanticscholar.org/3d77-/ced8e0078797b1f0bf54fac32b1f8984d74b.pdf. Acesso em: 20 dez. 2016.

GUO, H.; TRæTTEBERG, H.; WANG, A. I.; GAO, S. A workflow for model driven game development. In: **Enterprise Distributed Object Computing Conference (EDOC), 2015 IEEE 19th International**. Adelaide, Australia: IEEE Computer Society, 2015. p. 94–103. ISSN 1541-7719.

GUTIERREZ, L.; NIKOLAIDIS, I.; STROULIA, E.; GOUGLAS, S.; ROCKWELL, G.; BOECHLER, P.; CARBONARO, M.; KING, S. far-play: A framework to develop augmented/alternate reality games. In: **2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)**. Seattle, WA, USA: IEEE Computer Society, 2011. p. 531–536.

HENDRIKX, M.; MEIJER, S.; VELDEN, J. V. D.; IOSUP, A. Procedural content generation for games: A survey. **ACM Trans. Multimedia Comput. Commun. Appl.**, ACM, New York, NY, USA, v. 9, n. 1, p. 1:1–1:22, Feb. 2013. ISSN 1551-6857.

HOLM, J.; LAURILA, K. Designing actiontrack: A state-of-the-art authoring tool for location-based games and other activities. In: **2014 18th International Conference on Information Visualisation**. Paris, France: IEEE Computer Society, 2014. p. 402–407. ISSN 1550-6037.

IOSUP, A. Poggi: generating puzzle instances for online games on grid infrastructures. **Concurrency and Computation: Practice and Experience**, Wiley Online Library, *[s.l.]*, v. 23, n. 2, p. 158–171, 2011.

JACOB, J. T. P. N.; COELHO, A. F. Issues in the development of location-based games. **International Journal of Computer Games Technology**, Hindawi Publishing Corporation, *[s.l.]*, v. 2011, 2011.

JAFFE, A.; MILLER, A.; ANDERSEN, E.; LIU, Y.-E.; KARLIN, A.; POPOVIć, Z. Evaluating competitive game balance with restricted play. In: **Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment**. Stanford, California, USA: AAAI Press, 2012. (AIIDE'12), p. 26–31.

JAKL, A. The workflow of c++ game-development on a series 60 platform device. **FH Oberöstereich**, Citeseer, Hagenberg, Austria, 2004.

JORDAN, K. O.; SHEPTYKIN, I.; GRÜTER, B.; VATTERROTT, H.-R. Identification of structural landmarks in a park using movement data collected in a location-based game. **Proceedings of the 21st SIGSPATIAL International Conference on Advances in Geographic Information Systems**, ACM, Orlando, FL, USA, v. 13, 2013.

JUN, T. A survey on the bandit problem with switching costs. **De Economist**, Kluwer Academic Publishers, Dordrecht, Netherlands, v. 152, n. 4, p. 513–541, Dec 2004. ISSN 1572-9982.

JURGELIONIS, A.; WETZEL, R.; BLUM, L.; OPPERMANN, L. Totem.scout: A mobile tool for in-situ creation of location-based content. In: **2013 IEEE International Games Innovation Conference (IGIC)**. Vancouver, Canada: IEEE Computer Society, 2013. p. 89–96. ISSN 2166-6741.

KASAPAKIS, V.; GAVALAS, D. Pervasive gaming: Status, trends and design principles. **Journal of Network and Computer Applications**, Elsevier, *[s.l.]*, v. 55, p. 213–236, 2015.

KASAPAKIS, V.; GAVALAS, D.; BUBARIS, N. Addressing openness and portability in outdoor pervasive role-playing games. In: IEEE. **Communications and Information Technology (ICCIT), 2013 Third International Conference on**. Beirut, Lebanon: IEEE Computer Society, 2013. p. 93–97.

KHOO, K.; SUGANTHAN, P. Evaluation of genetic operators and solution representations for shape recognition by genetic algorithms. **Pattern Recognition Letters**, *[s.l.]*, v. 23, n. 13, p. 1589 – 1597, 2002. ISSN 0167-8655.

KIRMAN, B.; LINEHAN, C.; LAWSON, S. Blowtooth: a provocative pervasive game for smuggling virtual drugs through real airport security. **Personal and Ubiquitous Computing**, Springer-Verlag, *[s.l.]*, v. 16, n. 6, p. 767–775, 2012.

KOCSIS, L.; SZEPESVáRI, C. Bandit based monte-carlo planning. In: **Proceedings of the 17th European Conference on Machine Learning**. Berlin, Heidelberg: Springer-Verlag, 2006. (ECML'06), p. 282–293. ISBN 3-540-45375-X, 978-3-540-45375-8.

KOSTER, R. **A Theory of Fun for Game Design**. Scottsdale, Arizona: Paraglyph Press, 2005. 244 p.

KRCMAR, M.; DHAWAN, A. P. Application of genetic algorithms in graph matching. In: **Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)**. Orlando, FL, USA: IEEE Computer Society, 1994. v. 6, p. 3872–3876 vol.6.

LAATO., S.; PIETARINEN., T.; RAUTI., S.; PALOHEIMO., M.; INABA., N.; SUTINEN., E. A review of location-based games: Do they all support exercise, social interaction and cartographical training? In: INSTICC. **Proceedings of the 11th International Conference on Computer Supported Education - Volume 1: CSEDU,**. Heraklion, Crete, Greece: SciTePress, 2019. p. 616–627. ISBN 978-989-758-367-4.

LABS, N. **Polygonal Map Generation for Games**. 2019. Disponível em: http://www-cs-students.stanford.edu/˜amitp/game-programming/polygon-map-generation/. Acesso em: 20 dez. 2018.

LAJEVARDI, S. M.; ARAKALA, A.; DAVIS, S.; HORADAM, K. Retina verification system based on biometric graph matching. **IEEE Transactions on Image Processing**, *[s.l.]*, v. 22, n. 9, p. 3625–3635, Sep. 2013. ISSN 1057-7149.

LEHMANN, L. **Location-based Mobile Games**. Berlin, Germany: GRIN Verlag, 2012. Disponível em: https://www.snet.tu-berlin.de/fileadmin/fg220/courses/WS1112/snet-project-/location-based-mobile-games_lehmann.pdf. Acesso em: 20 dez. 2016.

LELIS, L. H. S.; REIS, W. M. P.; GAL, Y. Procedural generation of game maps with human-in-the-loop algorithms. **IEEE Transactions on Games**, IEEE Computer Society, *[s.l.]*, v. 10, n. 3, p. 271–280, Sep. 2018. ISSN 2475-1502.

LI, Y.-T.; WACHS, J. P. Hierarchical elastic graph matching for hand gesture recognition. In: ALVAREZ, L.; MEJAIL, M.; GOMEZ, L.; JACOBO, J. (Ed.). **Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 308–315. ISBN 978-3-642-33275-3.

LI, Z.; CHEN, B.; CHE, D. Solving the subgraph isomorphism problem using simulated annealing and evolutionary algorithms. In: THE STEERING COMMITTEE OF THE WORLD CONGRESS IN COMPUTER SCIENCE, COMPUTER ENGINEERING AND APPLIED COMPUTING (WORLDCOMP). **Proceedings on the International Conference on Artificial Intelligence (ICAI)**. Las Vegas, USA, 2016. p. 293–299.

LIEBERMAN, H.; PATERNÒ, F.; KLANN, M.; WULF, V. End-user development: An emerging paradigm. In: **End User Development**. Dordrecht: Springer Netherlands, 2006. p. 1–8. ISBN 978-1-4020-5386-3.

LIN, W.; XIAO, X.; CHENG, J.; BHOWMICK, S. S. Efficient algorithms for generalized subgraph query processing. In: **Proceedings of the 21st ACM International Conference on Information and Knowledge Management**. New York, NY, USA: ACM, 2012. (CIKM '12), p. 325–334. ISBN 978-1-4503-1156-4.

LITTLE, R. **The Pokemon Company, Nintendo and Google Put $30 Million into Pokemon GO Developer**. 2016. Disponível em: https://gamerant.com/pokemon-go-30-million-dollar-investment/. Acesso em: 20 fev. 2019.

LIU, C.-W.; FAN, K.-C.; HORNG, J.-T.; WANG, Y.-K. Solving weighted graph matching problem by modified microgenetic algorithm. In: **1995 IEEE International Conference on Systems, Man and Cybernetics. Intelligent Systems for the 21st Century**. Vancouver, Canada: IEEE Computer Society, 1995. v. 1, p. 638–643 vol.1.

LIVI, L.; RIZZI, A. The graph matching problem. **Pattern Anal. Appl.**, Springer-Verlag, London, UK, UK, v. 16, n. 3, p. 253–283, Aug. 2013. ISSN 1433-7541.

LOCHRIE, M.; PUCIHAR, K. C.; GRADINAR, A.; COULTON, P. Designing seamless mobile augmented reality location based game interfaces. In: **Proceedings of International Conference on Advances in Mobile Computing & Multimedia**. New York, NY, USA: ACM, 2013. (MoMM '13), p. 412:412–412:415. ISBN 978-1-4503-2106-8.

LUDIA. **Jurassic World Alive**. 2018. Disponível em: https://www.jurassicworldalive.com. Acesso em: 20 dez. 2018.

LUND, K.; COULTON, P.; WILSON, A. Participation inequality in mobile location games. In: ACM. **Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology**. Lisbon, Portugal: ACM, 2011. p. 27.

LUND, K.; LOCHRIE, M.; COULTON, P. Enabling emergent behaviour in location based games. In: ACM. **Proceedings of the 14th International Academic MindTrek Conference: Envisioning Future Media Environments**. Tampere, Finland: ACM, 2010. p. 78–85.

MACVEAN, A.; HAJARNIS, S.; HEADRICK, B.; FERGUSON, A.; BARVE, C.; KARNIK, D.; RIEDL, M. O. Wequest: scalable alternate reality games through end-user content authoring. In: ACM. **Proceedings of the 8th international conference on advances in computer entertainment technology**. Lisbon, Portugal: ACM, 2011. p. 22.

MAIA, L. F.; RODRIGUES, W.; VIANA, W.; TRINTA, F. Auragame: A case study of a zero programming augmented reality game. SBGames, São Paulo, Brazil, 2016.

MAIA, L. F.; VIANA, W.; TRINTA, F. Using monte carlo tree search and google maps to improve game balancing in location-based games. In: **2017 IEEE Conference on Computational Intelligence and Games (CIG)**. NY, NY, USA: IEEE Computer Society, 2017. p. 215–222. ISSN 2325-4289.

MALEGIANNAKI, I.; DARADOUMIS, T. Analyzing the educational design, use and effect of spatial games for cultural heritage: A literature review. **Computers & Education**, *[s.l.]*, v. 108, p. 1 – 10, 2017. ISSN 0360-1315.

MALONE, T. What makes computer games fun? (abstract only). In: **Proceedings of the Joint Conference on Easier and More Productive Use of Computer Systems. (Part - II): Human Interface and the User Interface - Volume 1981**. New York, NY, USA: ACM, 1981. (CHI '81), p. 143–. ISBN 0-89791-064-8.

MCGONIGAL, J. A real little game: The performance of belief in pervasive play. **Proceedings of DiGRA 2003**, Utrecht, Netherlands, 2003.

MCLAUGHL, M. **New GTA V release tipped to rake in £1bn in sales**. 2013. Disponível em: https://www.scotsman.com/lifestyle/gadgets-gaming/new-gta-v-release-tipped-to-rake-in-1bn-in-sales-1-3081943. Acesso em: 20 fev. 2019.

MISUND, G.; HOLONE, H.; KARLSEN, J.; TOLSBY, H. Chase and catch-simple as that?: old-fashioned fun of traditional playground games revitalized with location-aware mobile phones. In: ACM. **Proceedings of the International Conference on Advances in Computer Enterntainment Technology**. Athens, Greece: ACM, 2009. p. 73–80.

MOGHADAM, A. B.; RAFSANJANI, M. K. A genetic approach in procedural content generation for platformer games level creation. In: **2017 2nd Conference on Swarm**

Intelligence and Evolutionary Computation (CSIEC). Kerman, Iran: *[s.n.]*, 2017. p. 141–146.

NEUSTAEDTER, C.; JUDGE, T. K. See it: a scalable location-based game for promoting physical activity. In: ACM. **Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work Companion**. *[s.l.]*: ACM, 2012. p. 235–238.

NEVELSTEEN, K. J. L. Survey of pervasive games and technologies. In: **A Survey of Characteristic Engine Features for Technology-Sustained Pervasive Games**. Cham: Springer International Publishing, 2015. p. 11–39. ISBN 978-3-319-17632-1.

NEWZOO. **Mobile Revenues Account for More Than 50% of the Global Games Market as It Reaches $137.9 Billion in 2018**. 2018. Report. Disponível em: https://newzoo.com-/insights/articles/global-games-market-reaches-137-9-billion-in-2018-mobile-games-take-half/. Acesso em: 20 dez. 2018.

NEXT. **The Walking Dead Our World**. 2018. Disponível em: https://www-.thewalkingdeadourworld.com. Acesso em: 20 dez. 2018.

NIANTIC. **Pokémon Go**. 2016. Disponível em: http://www.pokemongo.com. Acesso em: 20 dez. 2018.

NICKLAS, D.; PFISTERER, C.; MITSCHANG, B. Towards location-based games. In: **Proceedings of the International Conference on Applications and Development of Computer Games in the 21st Century: ADCOG**. Hong Kong, China: *[s.n.]*, 2001. v. 21, p. 61–67.

NIEUWDORP, E. The pervasive discourse: an analysis. **Computers in Entertainment (CIE)**, ACM, *[s.l.]*, v. 5, n. 2, p. 13, 2007.

NOLETO, C.; LIMA, M.; MAIA, L. F.; VIANA, W.; TRINTA, F. An authoring tool for location-based mobile games with augmented reality features. In: **Computer Games and Digital Entertainment (SBGAMES), 2015 Brazilian Symposium on**. Teresina, Brazil: *[s.n.]*, 2015.

O'HARA, K. Understanding geocaching practices and motivations. In: **Proceedings of the SIGCHI Conference on Human Factors in Computing Systems**. New York, NY, USA: ACM, 2008. (CHI '08), p. 1177–1186. ISBN 978-1-60558-011-1.

OLESEN, J. K.; YANNAKAKIS, G. N.; HALLAM, J. Real-time challenge balance in an rts game using rtneat. In: **2008 IEEE Symposium On Computational Intelligence and Games**. Perth, Australia: IEEE Computer Society, 2008. p. 87–94. ISSN 2325-4270.

PAAVILAINEN, J.; KORHONEN, H.; ALHA, K.; STENROS, J.; KOSKINEN, E.; MAYRA, F. The pokémon go experience: A location-based augmented reality mobile game goes mainstream. In: **Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems**. New York, NY, USA: ACM, 2017. (CHI '17), p. 2493–2498. ISBN 978-1-4503-4655-9.

PARISH, Y. I. H.; MüLLER, P. Procedural modeling of cities. In: **Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques**. New York, NY, USA: ACM, 2001. (SIGGRAPH '01), p. 301–308. ISBN 1-58113-374-X.

PATEL, A. **Ingress**. 2016. Disponível em: http://www.ingress.com. Acesso em: 20 dez. 2018.

PATRO, A.; RAYANCHU, S.; GRIEPENTROG, M.; MA, Y.; BANERJEE, S. The anatomy of a large mobile massively multiplayer online game. **ACM SIGCOMM Computer Communication Review**, ACM, *[s.l.]*, v. 42, n. 4, p. 479–484, 2012.

PEDERSEN, C.; TOGELIUS, J.; YANNAKAKIS, G. N. Modeling player experience for content creation. **IEEE Transactions on Computational Intelligence and AI in Games**, *[s.l.]*, v. 2, n. 1, p. 54–67, March 2010. ISSN 1943-068X.

POTVIN, J.-Y. Genetic algorithms for the traveling salesman problem. **Annals of Operations Research**, *[s.l.]*, v. 63, n. 3, p. 337–370, Jun 1996. ISSN 1572-9338.

PROCYK, J.; NEUSTAEDTER, C. Gems: the design and evaluation of a location-based storytelling game. In: ACM. **Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing**. Baltimore, USA: ACM, 2014. p. 1156–1166.

RAVEAUX, R.; BURIE, J.-C.; OGIER, J.-M. A graph matching method and a graph matching distance based on subgraph assignments. **Pattern Recognition Letters**, Elsevier Science Inc., *[s.l.]*, v. 31, n. 5, p. 394 – 406, 2010. ISSN 0167-8655.

REID, J. Design for coincidence: Incorporating real world artifacts in location based games. In: **Proceedings of the 3rd International Conference on Digital Interactive Media in Entertainment and Arts**. New York, NY, USA: ACM, 2008. (DIMEA '08), p. 18–25. ISBN 978-1-60558-248-1.

REIS, W. M. P.; LELIS, L. H. S.; GAL, Y. K. Human computation for procedural content generation in platform games. In: **2015 IEEE Conference on Computational Inteligence in Games (CIG)**. Tainan, Taiwan: IEEE Computer Society, 2015. p. 99–106. ISBN 978-1-4799-8622-4.

ROUNGAS, B.; DALPIAZ, F. A model-driven framework for educational game design. In: SPRINGER. **International Conference on Games and Learning Alliance**. Rome, Italy: Springer, 2015. p. 1–11.

RUBINO, I.; BARBERIS, C.; XHEMBULLA, J.; MALNATI, G. Integrating a location-based mobile game in the museum visit: Evaluating visitors' behaviour and learning. **Journal on Computing and Cultural Heritage (JOCCH)**, ACM, *[s.l.]*, v. 8, n. 3, p. 15, 2015.

SANFELIU, A.; FU, K. A distance measure between attributed relational graphs for pattern recognition. **IEEE Transactions on Systems, Man, and Cybernetics**, *[s.l.]*, SMC-13, n. 3, p. 353–362, May 1983. ISSN 0018-9472.

SANTOS, O. L. dos; RAFALSKI, J. do P.; MENEZES, C. S. de. Uma game engine para aventuras pedagógicas locativas em realidade aumentada. In: **Anais do Simpósio Brasileiro de Informática na Educação**. Campinas, Brasil: *[s.n.]*, 2013. v. 24, n. 1, p. 396.

SCHELL, J. **The Art of Game Design: A Book of Lenses**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008. ISBN 0-12-369496-5.

SCHILLE, J. **$44 Million For God Of War 3**. 2010. Disponível em: https://gamerant.com/44-million-god-war-3/. Acesso em: 20 fev. 2019.

SERAPH, L. O. **BattleSuit Runner Fitness**. 2016. Disponível em: http://apps.appshout.com-/battlesuit-runner-fitness/. Acesso em: 20 dez. 2018.

SILVA, A. de Souza e; SUTKO, D. **Digital Cityscapes: Merging Digital and Urban Playspaces**. Bern, Switzerland: Peter Lang, 2009. ISBN 9781433105326. Disponível em: https://books.google.com.br/books?id=OfUDGUI_GXgC. Acesso em: 20 jan. 2016.

SILVA, L. F. M.; NOLETO, C.; LIMA, M.; FERREIRA, C.; MARINHO, C.; VIANA, W.; TRINTA, F. LAGARTO: A location based games authoring tool enhanced with augmented reality features. **Entertainment Computing**, Elsevier Science Inc., *[s.l.]*, v. 22, p. 3–13, 2017.

SINGH, M.; CHATTERJEE, A.; CHAUDHURY, S. Matching structural shape descriptions using genetic algorithms. **Pattern Recognition**, Elsevier Science Inc., *[s.l.]*, v. 30, n. 9, p. 1451 – 1462, 1997. ISSN 0031-3203.

SMITH, A. R. Plants, fractals, and formal languages. In: **Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques**. New York, NY, USA: ACM, 1984. (SIGGRAPH '84), p. 1–10. ISBN 0-89791-138-5.

SMITH, G.; TREANOR, M.; WHITEHEAD, J.; MATEAS, M. Rhythm-based level generation for 2d platformers. In: **Proceedings of the 4th International Conference on Foundations of Digital Games**. New York, NY, USA: ACM, 2009. (FDG '09), p. 175–182. ISBN 978-1-60558-437-9.

SMITH, T.; PADGET, J.; VIDLER, A. Graph-based generation of action-adventure dungeon levels using answer set programming. In: **Proceedings of the 13th International Conference on the Foundations of Digital Games**. New York, NY, USA: ACM, 2018. (FDG '18), p. 52:1–52:10. ISBN 978-1-4503-6571-0.

SNODGRASS, S.; ONTAÑÓN, S. A hierarchical approach to generating maps using markov chains. In: **Proceedings of the Tenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment**. Raleigh, North Carolina, USA: AAAI Press, 2014. (AIIDE'14), p. 59–65. ISBN 1577356810, 978-1-57735-681-3.

SNODGRASS, S.; ONTAÑÓN, S. Learning to generate video game maps using markov models. **IEEE Transactions on Computational Intelligence and AI in Games**, IEEE Computer Society, *[s.l.]*, v. 9, n. 4, p. 410–422, Dec 2017. ISSN 1943-068X.

SÖBKE, H.; HAUGE, J. B.; STEFAN, I. A. Prime Example Ingress Reframing the Pervasive Game Design Framework (PGDF). **International Journal of Serious Games**, Serious Games Society, Italy, v. 4, n. 2, p. 39–58, Jun. 2017.

SORENSON, N.; PASQUIER, P. Towards a generic framework for automated video game level creation. In: CHIO, C. D.; CAGNONI, S.; COTTA, C.; EBNER, M.; EKÁRT, A.; ESPARCIA-ALCAZAR, A. I.; GOH, C.-K.; MERELO, J. J.; NERI, F.; PREUSS, M.; TOGELIUS, J.; YANNAKAKIS, G. N. (Ed.). **Applications of Evolutionary Computation**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 131–140. ISBN 978-3-642-12239-2.

SPIESBERGER, P.; JUNGWIRTH, F.; WÖSS, C.; BACHL, S.; HARMS, J.; GRECHENIG, T. Woody: a location-based smartphone game to increase children's outdoor activities in urban environments. In: ACM. **Proceedings of the 14th International Conference on Mobile and Ubiquitous Multimedia**. Linz, Austria: ACM, 2015. p. 368–372.

STANLEY, K. G.; LIVINGSTON, I.; BANDURKA, A.; KAPISZKA, R.; MANDRYK, R. L. Pinizoro: a gps-based exercise game for families. In: ACM. **Proceedings of the International Academic Conference on the Future of Game Design and Technology**. Vancouver, Canada: ACM, 2010. p. 243–246.

START, S. to; ALDERMAN, N. **The Walk**. 2016. Disponível em: http://www.thewalkgame.com. Acesso em: 20 dez. 2018.

SUN, J.; YU, X.; BACIU, G.; GREEN, M. Template-based generation of road networks for virtual city modeling. In: **Proceedings of the ACM Symposium on Virtual Reality Software and Technology**. New York, NY, USA: ACM, 2002. (VRST '02), p. 33–40. ISBN 1-58113-530-0.

SUPERDATA. **3 reasons why mobile AR developers should focus on ads and not in-app purchases**. 2018. Disponível em: https://www.superdataresearch.com/3-reasons-why-mobile-ar-developers-should-focus-on-ads-and-not-in-app-purchases/. Acesso em: 20 dez. 2018.

TOGELIUS, J.; KASTBJERG, E.; SCHEDL, D.; YANNAKAKIS, G. N. What is procedural content generation?: Mario on the borderline. In: **Proceedings of the 2Nd International Workshop on Procedural Content Generation in Games**. New York, NY, USA: ACM, 2011. (PCGames '11), p. 3:1–3:6. ISBN 978-1-4503-0872-4.

TOGELIUS, J.; PREUSS, M.; BEUME, N.; WESSING, S.; HAGELBÄCK, J.; YANNAKAKIS, G. N. Multiobjective exploration of the starcraft map space. In: **Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games**. Copenhagen, Denmark: IEEE Computer Society, 2010. p. 265–272. ISSN 2325-4270.

TRAN, H.-N.; CAMBRIA, E.; HUSSAIN, A. Towards gpu-based common-sense reasoning: Using fast subgraph matching. **Cognitive Computation**, Springer US, *[s.l.]*, v. 8, n. 6, p. 1074–1086, Dec 2016. ISSN 1866-9964.

TREGEL, T.; RAYMANN, L.; GÖBEL, S.; STEINMETZ, R. Geodata classification for automatic content creation in location-based games. In: ALCAÑIZ, M.; GÖBEL, S.; MA, M.; OLIVEIRA, M. F.; HAUGE, J. B.; MARSH, T. (Ed.). **Serious Games**. Cham: Springer International Publishing, 2017. p. 212–223. ISBN 978-3-319-70111-0.

ULLMANN, J. R. An algorithm for subgraph isomorphism. **J. ACM**, ACM, New York, NY, USA, v. 23, n. 1, p. 31–42, Jan. 1976. ISSN 0004-5411.

UMEYAMA, S. An eigendecomposition approach to weighted graph matching problems. **IEEE Trans. Pattern Anal. Mach. Intell.**, IEEE Computer Society, Washington, DC, USA, v. 10, n. 5, p. 695–703, Sep. 1988. ISSN 0162-8828.

VALENTE, L.; FEIJÓ, B. Extending use cases to support activity design in pervasive mobile games. In: **2014 Brazilian Symposium on Computer Games and Digital Entertainment**. Porto Alegre, Brazil: *[s.n.]*, 2014. p. 193–201. ISSN 2159-6654.

VALENTE, L.; FEIJÓ, B.; LEITE, J. C. S. d. P. Mapping quality requirements for pervasive mobile games. **Requirements Engineering**, *[s.l.]*, v. 22, n. 1, p. 137–165, Mar 2017. ISSN 1432-010X.

VALENTE, L.; FEIJÓ, B.; LEITE, J. C. S. d. P.; CLUA, E. A method to assess pervasive qualities in mobile games. **Personal and Ubiquitous Computing**, *[s.l.]*, v. 22, n. 4, p. 647–670, Aug 2018. ISSN 1617-4917.

VIANA, J. R. M.; VIANA, N. P.; TRINTA, F. A. M.; CARVALHO, W. V. d. A systematic review on software engineering in pervasive games development. In: **2014 Brazilian Symposium on Computer Games and Digital Entertainment**. Porto Alegre, Brazil: *[s.n.]*, 2014. p. 51–60. ISSN 2159-6662.

WAKE, J. D. **Mobile, location-based games for learning: Developing, deploying and evaluating mobile game technology in education**. Phd Thesis (PhD Thesis) — The University of Bergen, Bergen, Norway, 2013. Disponível em: http://hdl.handle.net/1956/7376.

WEISER, M. The computer for the 21st century. **Scientific american**, Nature Publishing Group, *[s.l.]*, v. 265, n. 3, p. 94–104, 1991.

WESTRA, J.; HASSELT, H. van; DIGNUM, V.; DIGNUM, F. On-line adapting games using agent organizations. In: **2008 IEEE Symposium On Computational Intelligence and Games**. Perth, Australia: IEEE Computer Society, 2008. p. 243–250. ISSN 2325-4270.

WETZEL, R.; BLUM, L.; OPPERMANN, L. Tidy city: a location-based game supported by in-situ and web-based authoring tools to enable user-created content. In: ACM. **Proceedings of the International Conference on the Foundations of Digital Games**. Raleigh, NC, USA: ACM, 2012. p. 238–241.

WITKOWSKI, E. Running from zombies. In: **Proceedings of The 9th Australasian Conference on Interactive Entertainment: Matters of Life and Death**. New York, NY: ACM, 2013. p. 1.

WORKLINE, C. **Tourality – The Ultimate Outdoor GPS Game**. 2016. Disponível em: http://www.tourality.com. Acesso em: 12 nov. 2016.

XIANG, Y.; HAN, J.; XU, H.; GUO, X. An improved heuristic method for subgraph isomorphism problem. **IOP Conference Series: Materials Science and Engineering**, IOP Publishing, Singapore, v. 231, p. 012050, sep 2017.

XIUTANG, G.; KAI, Z. Simulated annealing algorithm for detecting graph isomorphism*. **Journal of Systems Engineering and Electronics**, Elsevier Science Inc., *[s.l.]*, v. 19, n. 5, p. 1047 – 1052, 2008. ISSN 1004-4132.

XU, Y.; SALAPAKA, S. M.; BECK, C. L. A distance metric between directed weighted graphs. In: **52nd IEEE Conference on Decision and Control**. Firenze, Italy: IEEE Computer Society, 2013. p. 6359–6364. ISSN 0191-2216.

YE, M.; YIN, P.; LEE, W.-C.; LEE, D.-L. Exploiting geographical influence for collaborative point-of-interest recommendation. In: **Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval**. New York, NY, USA: ACM, 2011. (SIGIR '11), p. 325–334. ISBN 978-1-4503-0757-4.

ZENDER, R.; METZLER, R.; LUCKE, U. Freshup—a pervasive educational game for freshmen. **Pervasive and Mobile Computing**, *[s.l.]*, v. 14, p. 47 – 56, 2014. ISSN 1574-1192. Special Issue on Pervasive EducationSpecial Issue on The Social Car: Socially-inspired Mechanisms for Future Mobility Services.

APPENDIX A – USER QUESTIONNAIRE

Question 1. Name:

Question 2. Age:

Question 3. Have you ever played a LBG?

(a) Yes, and I play frequently.

(b) Yes but only for a short period.

(c) No, but I do know how they work.

(d) No, and I don't know how they work.

Question 4. Do you think that current LBGs can be played everywhere?

(a) Yes, I totally agree.

(b) Yes, I partially agree.

(c) No, I partially disagree.

(d) No, I totally disagree.

Question 5. Do you think that players have advantages depending on their location in LBGs?

(a) Yes, I totally agree.

(b) Yes, I partially agree.

(c) No, I partially disagree.

(d) No, I totally disagree.

APPENDIX B – KERNELS USED IN THE PWU

Source-code 1 – Parallel Weighted Ullman

```
1
2  __kernel void weighted_ullman(__global const int* P,
       __global const int* G, __global const int* params,
       __global const int* permutations, __global int* res){
3    unsigned int thread_id=get_global_id(0);
4    int row_size=params[0];
5    int col_size=params[1];
6
7    int indexes[10];
8    for (int i = 0; i < row_size; ++i) {
9      indexes[i]=permutations[(thread_id*row_size)+i];
10   }
11
12   int diff = 0;
13   int i_g = 0;
14   int j_g = 0;
15   for (int i = 0; i < row_size; ++i) {
16     i_g = indexes[i];
17     for (int j = 0; j < row_size; ++j) {
18       j_g = indexes[j];
19       if ((i!=j) && (P[i * row_size + j] != -1)) {
20         diff += abs(P[i * row_size + j] - G[i_g * col_size
               + j_g]);
21       }
22     }
23   }
24
25   res[thread_id]=diff;
26   barrier(CLK_LOCAL_MEM_FENCE);
```

```
27 }
```

Source-code 2 – Create Root Matrix

```
1  static int insert_unique(int* match_col, int size, int row,
       int col) {
2    for (int i = 0; i < size; i += 2) {
3      if ((match_col[i] == row) && (match_col[i + 1] == col))
           {
4        return size;
5      }
6    }
7    match_col[size] = row;
8    match_col[size + 1] = col;
9    return (size + 2);
10 }
11
12 __kernel void create_root_matrix(__global const int* P,
     __global const int* G, __global const int* grades_P,
     __global const int* grades_G, __global const float*
     threshold_matrix, __global const int* params, __global
     float* res){
13
14   unsigned int thread_id=get_global_id(0);
15   int size_P=params[0];
16   int size_G=params[1];
17   int match_col[50*2];
18   int row_p=(thread_id/size_G);
19   int row_g=thread_id%size_G;
20
21   if(grades_P[row_p] > grades_G[row_g]) {
22     res[thread_id]=0;
```

```
23  } else {
24
25    int pos=0;
26    int hasMatch=0;
27    for (int col_p=0; col_p<size_P;col_p++) {
28      int p_x = P[(row_p*size_P)+col_p];
29      if((col_p == row_p) || (p_x == -1)) {
30        continue;
31      }
32
33      hasMatch=0;
34      for (int col_g=0; col_g<size_G;col_g++) {
35        int g_x = G[(row_g*size_G)+col_g];
36        if((col_g == row_g) || (g_x == -1)) {
37          continue;
38        }
39        float res = abs(p_x - g_x);
40        if(res <= p_x * threshold_matrix[(row_p*size_P)+
             col_p]) {
41          pos=insert_unique(match_col,pos, row_g, col_g);
42          hasMatch=1;
43        }
44      }
45      if(hasMatch==0) {
46        break;
47      }
48    }
49
50    if((pos==0)||(grades_P[row_p] > (pos/2))||(hasMatch==0)
       ) {
51      res[thread_id]=0;
52    } else {
```

```
53        res[thread_id]=1;
54      }
55    }
56
57    barrier(CLK_LOCAL_MEM_FENCE);
58
59  }
```

Source-code 3 – Parallel Brute Force

```
1  static void calculate_diffs(int* remaining, const int* nums
       , const int* indexes, int row_size, int col_size) {
2
3    for (int i = 0; i < col_size; i++) {
4      remaining[i] = i;
5    }
6
7    int pos = 0;
8    for (int i = 0; i < col_size; i++) {
9      bool selected = false;
10     for (int j = 0; j < row_size; j++) {
11       if (indexes[j] == -1) {
12         break;
13       }
14       if (nums[i] == indexes[j]) {
15         selected = true;
16         break;
17       }
18     }
19     if (selected == false) {
20       remaining[pos] = nums[i];
21       pos++;
```

```
22      }
23    }
24  }
25
26  static void decodeIndexes(int* indexes, int id, int
       row_size, int col_size, int total) {
27    int partitions[10];
28    partitions[0] = total;
29
30    int nums[50];
31    int remaining[50];
32    for (int i = 0; i < col_size; i++) {
33      nums[i] = i;
34    }
35
36    for (int i = 0; i < row_size; i++) {
37      indexes[i] = -1;
38    }
39
40    int pos = 0;
41    for (int i = 0; i < row_size; i++) {
42      partitions[i + 1] = (partitions[i] / (col_size - i));
43      pos = ((id % partitions[i]) / partitions[i + 1]);
44      calculate_diffs(remaining, nums, indexes, row_size,
           col_size);
45      indexes[i] = remaining[pos];
46    }
47  }
48
49  __kernel void brute_force(__global const int* P, __global
       const int* G, __global const int* params, __global int*
       res){
```

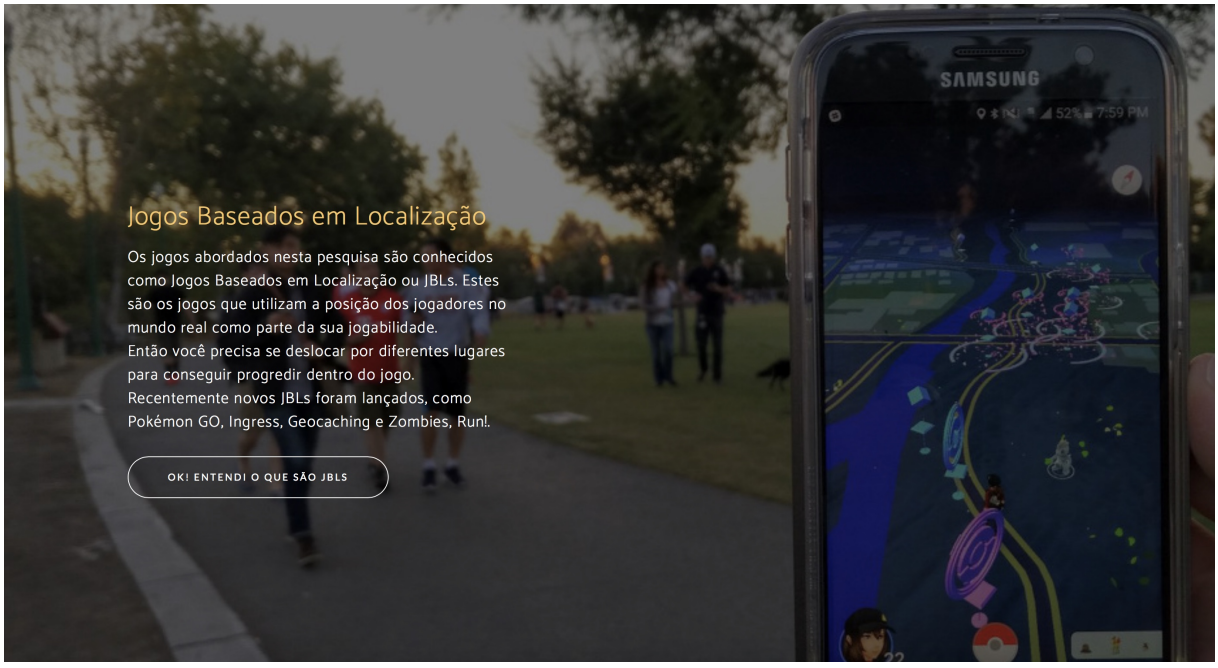
```
50    unsigned int thread_id=get_global_id(0);
51    int row_size=params[0];
52    int col_size=params[1];
53    int total_permutations=params[2];
54
55    int indexes[10];
56    decodeIndexes(indexes,thread_id,row_size,col_size,
          total_permutations);
57
58    int diff = 0;
59    int i_g = 0;
60    int j_g = 0;
61    for (int i = 0; i < row_size; ++i) {
62      i_g = indexes[i];
63      for (int j = 0; j < row_size; ++j) {
64        j_g = indexes[j];
65        if ((i!=j) && (P[i * row_size + j] != -1)) {
66          diff += abs(P[i * row_size + j] - G[i_g * col_size
              + j_g]);
67        }
68      }
69    }
70    res[thread_id]=diff;
71    barrier(CLK_LOCAL_MEM_FENCE);
72  }
```

APPENDIX C – WEBSITE DEVELOPED TO THE EVALUATION

Figure 67 – Website's welcome page.

Source: Author

Figure 68 – Page defining Location-based Games.



Source: Author

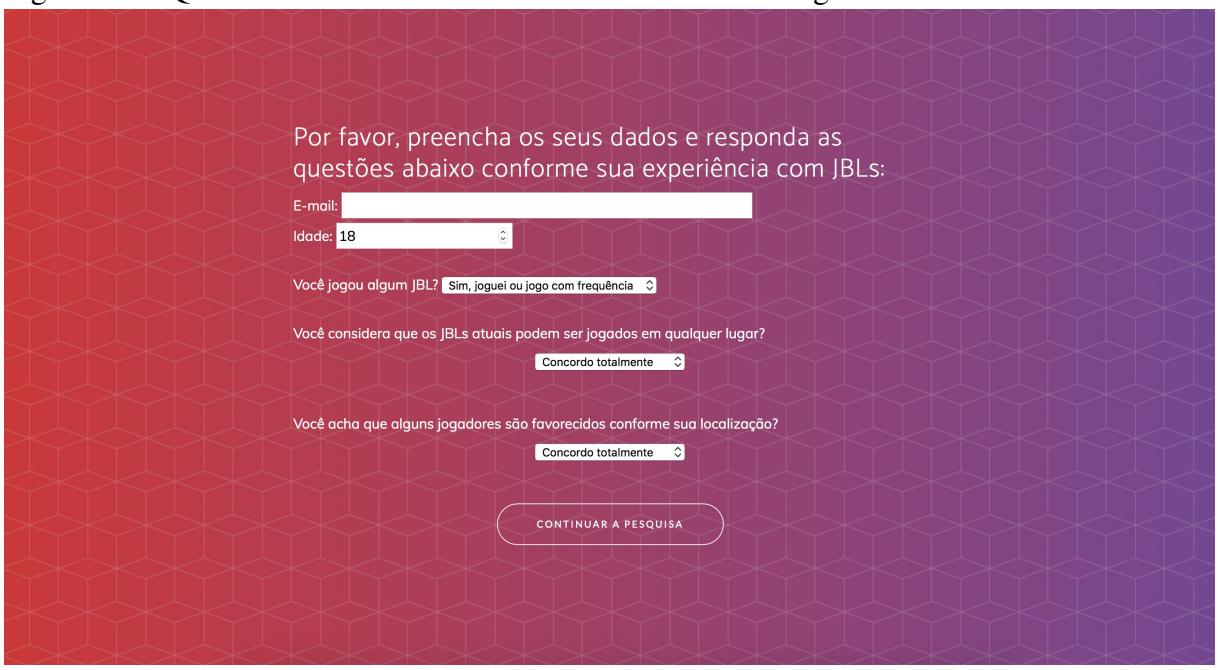
Figure 69 – In this page the challenges of this research were explained.

Source: Author

Figure 70 – Page containing a video that provides an overview of this research.

Source: Author

Figure 71 – Questionnaire to collect data about users' knowledge on LBGs.



Source: Author

Figure 72 – Page used to select locations where the maps of LBGs must be transposed to.

Source: Author