Instituto Tecnologico y de Estudios Superiores de Monterrey

Monterrey Campus

School of Engineering and Sciences

Videogame Crowdsourcing Approach to Find Strategies Using Repeated Sub-Sequences

A thesis presented by

Arturo Silva Gálvez

Submitted to the
School of Engineering and Sciences
in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science

Monterrey, Nuevo León, June, 2020

Dedication

Dedicated to my family, my inspiration for doing this research.

Acknowledgements

Firstly, I would like to express my gratitude to my thesis advisors Dr. Raúl Monroy Borja and Dr. José Emmanuel Ramírez Márquez, whose guidance allowed me to write this dissertation. I am very grateful for their advice and all the opportunities they gave me.

Thanks to the thesis committee members, Dr. Rene Mac Kinney Romero and Dr. Miguel Angel Medina Pérez, for all their comments. Their advice allowed me to notice the things my research missed and learn to avoid them in future works.

I would like to thank the members of the research group of Machine Learning that were involved in the process of this thesis, whose suggestions helped me to make the study.

To the Tecnológico de Monterrey and CONACyT, who supported me and made this research possible.

Thanks to my family for always being with me. To my parents Arturo and Ana Laura, for their encouragement and support. To my brothers Rodrigo and Ana Laura, for being along with me in this process. To my grandmothers María Luisa and Josefina, for their unconditional love and support. To my uncles and cousins that shared experiences of this research with me. And to my friends that always support me.

I want to give a special thanks to my friend Ricardo de la Barrera, who's not just always with me, but also helped me understand concepts that were essential for this research.

To my classmates, who shared many experiences with me during our time in the master's program. Especially to Erick Lara and Jorge Orozco, with whom I was fortunate to work more closely. They taught me many things that helped me grow as a professional and as an individual.

To Dr. José Carlos Ortíz Bayliss, who supported me with part of this work and shared knowledge that helped me become a better researcher.

To the director of the master's program, Dr. Hugo Terashima Marín, who trusted me and encouraged me in difficult times through this stage of my life.

And to all the players who supported this research with a match in the videogame.

Videogame Crowdsourcing Approach to Find Strategies Using Repeated Sub-Sequences

by
Arturo Silva Gálvez

Abstract

Crowdsourcing surged as a new problem-solving model to better knowledge on how to solve a specific problem. The procedure starts by externalizing the problem from the group that is trying to solve it. Then, people with a variety of skills can help design solutions. The motivation for persons to participate is the key that makes the model work. From giving money to socialization, many options exist to encourage people to contribute to a crowdsourcing model.

Studies tested the use of videogames to motivate people to participate in the solution to problems from different domains. These studies report that people can provide competitive solutions, against the experts, even for complex problems. Until now, Videogame Crowdsourcing helped to complement the solution space, but mining the strategies from the users is an area of opportunity.

This thesis studies the application of Videogame Crowdsourcing for mining strategies from players' solutions for a problem. It focuses on a specific one: the Housing Development Problem; it is of interest to the architecture community. It is a single objective problem that consists of placing as many houses as possible, given the land, subject to restrictions of connectivity (from the entrance of the land to all houses).

We represented a match of our videogame as a sequence of movements. Each move consists of placing a house on a square of the land, represented as a grid, followed by displacement to another square in which the player puts the next one. This representation abstracts out two types of plays: the ones made to fulfill the restrictions of connectivity and the ones that belong to a correction of a previous one. Our underlying hypothesis is that a player strategy lies within a grammar expression; in particular, it is embedded in the recurrent sub-sequences of the expression.

We used the videogame to collect 113 matches. With Sequitur, we found recurrent sub-sequences for each match, a larger sequence. Analyzing the sub-sequences, we have successfully identified the following strategies: Bottom-Left, Top-Right, Top-Left, and others that are not found as heuristics for optimization problems like the one on the videogame. Our results show that the strategy of a player is in the grammar expression of his/her movements. They encourage us to think that recurrent sub-sequences can build the strategies people use for the Housing Development Problem and lead to new algorithms.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Crowdsourcing emerged as a problem-solving model that tries to take advantage of the knowledge of a group of people to solve a task [7]. In it, people over the internet, with a variety of skills, help in the process of problem-solving. From collecting photographs of a theme to solving videogames, people use crowdsourced models in different areas. Depending on the objective of the designer, the model challenges people with distinct instances; from all the solutions generated from the model, experts design techniques to distinguish high-quality ideas in them.

One of the aspects that crowdsourcing models consider is the motivation for people to participate in them. They can help persons develop a skill, give money, or any other option that encourages someone to work in their task. Recent studies have shown interest in using videogames as motivation [17, 32, 44, 58]. Examples of the problems they tackle are labeling red blood cell images [44], or obtaining solutions for the robust facility location problem [17]. They report that people can provide competitive solutions, against experts, even in complex problems. The most recent study [58] models the behavior of people using a decision tree. Their approach uses all the matches they collected to train the decision tree; they do not divide the solutions to distinguish the best behaviors. In their research, they define a decision tree as a strategy.

In this thesis, we study the application of Videogame Crowdsourcing with the purpose of mining strategies from players' solutions for problems. We focus on the Housing Development Problem; it is of interest to the architecture community. Its objective is to place as many houses as possible in a given land, subject to restrictions of connectivity. The problem is a specific case of the Architectural Layout Design [75, 76]. It consists of finding the best locations and dimensions of a set of interrelated objects in the architecture field [47]. A motivation to solve the Housing Development Problem is to supply people with inexpensive housing in walkable neighborhoods [36]. The application of Videogame Crowdsourcing we created includes two components. First, a web platform that has all the interface of the videogame. We designed the videogame as a model of the Housing Development Problem [43, 49]. Second, a database that stores all the information about a match. This information includes all the plays and the final score.

Mining a player's strategy from a match is not straightforward; it includes all the decisions made to achieve the final goal [4]. Extracting strategies for a problem can guide future research into creating hopefully more powerful algorithms using player strategies [58]. Given all the options that a player has in the videogame, the task of analyzing them is a hard problem. To reduce the plays' search space, we abstracted much of the details to build a new representation. It consists of a sequence that describes the actions of the player during his/her match.

Our underlying hypothesis is that the strategy of a player lies in the recurrent sub-sequences of actions in the representation we designed. To find strategies, we need an algorithm capable of extracting the sub-sequences. One with this function is Sequitur [52]; it extends the alphabet of a sequence that uses one grammar with new rules. Each of them defines a sub-sequence of the original Sequitur received.

We collected and classified 113 matches with our application of Videogame Crowdsourcing. We divided all of them into positive and negative according to their score. The classification allowed us to find what differentiates the best players from the rest. We performed two experiments in the study; each of them applies Sequitur to get repeated sub-sequences. The first one validates the methodology with synthetic matches that use a similar strategy. The second experiment tests it with real players. From the real players, we identified several strategies, some of them existing in the literature and others we named in the study. The results suggest that algorithms can use recurrent sub-sequences to solve the Housing Development Problem.

In the following sections of this chapter, we talk about the motivation of this study and the problem definition. Here, we speak of the Housing Development Problem and crowdsourcing. Also, we state the hypothesis and objectives of this study. Then, we briefly show the solution approach for the problem and the main contributions of the research. At last, we describe the content of the next chapters.

1.1 Motivation

Housing is a basic need for humans. A place to live is something that all societies value. This concept can go from a house for an individual to somewhere a community lives. Buying or renting a home is a costly investment that families do [54]. However, it provides many benefits like security or privacy.

In high-demand areas at some places in the United States [21] or cities like Hong Kong [25], there is not enough affordable multifamily housing to meet the demand. However, this problem does not depend only on the land supply [25]. Building residential areas involve several aspects of urban planning and design, production cost, construction law, lack of infrastructure, municipal allocation, taxation, financing, among others [22].

The job of an architect is to assure that the environment in which people live satisfies their needs. Also, to use the land as best as possible. When residents are satisfied with their living place, their quality of life improves [18]. This quantifier can serve to the architect as an evaluation of the housing project developed for the people. To create a living environment, the architect must analyze different features. Some examples are the number of houses, access to the different zones of the complex, and other things that can come into their mind.

The problem of creating a living environment for people to live in is known as the Housing Development Problem. Recent studies showed interest in different aspects of this problem, like satisfaction or environmental issues [43, 49]. Other studies, more related to ours, evaluated different ways of housing affordability [36]. There, the results suggest that increasing the development of moderately priced houses will be affordable over time. What drives these studies is the goal of supplying people with inexpensive housing in walkable neighborhoods.

A generalization of the Housing Development Problem is the Spatial Configuration; it is a problem all physical designs must face. It consists of finding the best locations and dimensions of a set of interrelated objects that give the best performance to the design [47]. Examples of this optimization objective include Component Packing, Route Path Planning, Process and Facilities Layout Design, and Architectural Layout Design. The goal of the Architecture Layout Design is to find the best location for places in a building restricted to certain conditions [71]; the Housing Development Problem is similar, but its objective involves the objects in a residential complex.

There are many ways to model the Architectural Layout Design Problem. One way is to define the available space as a set of a grid with squares [47]. Another technique is to divide the problem into two parts [47]: the first one is the topology, which refers to the logical relations that the components have; the second one is the geometry, which defines its position and size.

The study of Liggett [35] proposed a method for space layout planning. It used a combined initial constructive placement with the improvement procedures. The first part finds an initial solution while the second one improves it by a pairwise technique. However, this algorithm can fall at a local optimum; it does not guaranty a global optimum. Another method employed to solve the architecture layout design problem is evolutionary algorithms [71]. More recent approaches applied simulated annealing [76] or gradient base methods [75] for multi-objective optimization of an Architectural Layout Design Problem.

Algorithms that compute solutions for an Architecture Layout Design Problem need to consider the representation of the objects. Michalek et al. [47] propose an optimization algorithm using geometric optimization to find the best location and size for a group of rectangles related between them. The objective is to place all the rooms of a building maximizing certain conditions. The model defines a $Unit$ as a rectangular space located for a specific architectural function, like rooms or a building. Each $Unit$ has a coordinate (x, y), which represents the object in space. The position of the four walls consists of the distances $[N, S, E$, and $W]$ to this coordinate. This representation has the advantage that all the walls can change independently, allowing a more flexible design strategy for optimization.

In this study, we aim to use a model for this problem to create a Videogame Crowdsourcing. Here we will focus on one of the many aspects a Housing Development Problem needs to optimize, the area occupied by residential houses. The goal is to abstract the spatial configuration problems that architects face when they try to design a residential area. Finding strategies for this problem can guide future research into creating algorithms using player strategies for Housing Development Problems that can be applied to similar Architecture Layout Design Problems.

1.2 Problem definition

A videogame is an electronic platform that allows a person to participate in a problem from another reality. Some of the reasons that can lead a person to play a videogame are that it helps them relieve stress, spend a moment that can be boring like time in a waiting room, and they can boost their creativity. A videogame can take care of the motivation when using crowdsourcing as a method to obtain solutions. Applications like speech processing, music generation, robotics, knowledge acquisition, and some others, use this strategy to solve problems in their area.

The growth of videogames attracted researchers to study the behavior of people in them [12, 4]. Several investigations have used Videogame Crowdsourcing with different objectives [17, 28, 32, 44, 58]. There are videogames for health issues [44, 32], data collection [67], optimization problems [17], among others. L. Perez et al. [17] used this approach to tackle the Refugee Aid Problem Policy. The study concludes that people can provide solutions that outperform algorithms in specific aspects. Other studies showed how people are capable of getting solutions competitive against experts [44]. All of them aimed to extract specific information about the solutions; more recent approaches extract Human-Derived-Heuristics or strategies using decision trees [28, 58].

The problem of player modeling is not new. Research in this area aims to generate algorithms capable of following players' behavior. Among the different models to achieve this objective, one area focus on recreating players' strategies [4]. The approaches more related to our work define a strategy with a set of rules and use machine learning methods to distinguish which of them allow the model to mimic the players' behavior. The experience of the researchers can aid them to predefine those rules. Another option is to use a structure the videogame has, like a tree of habilities or the map, to find rules. Approaches that exist in the literature use evolutionary algorithms [19], genetic algorithms [39], tree structures [53, 58], and Bayesian models [60] to analyze the information of a set of matches and find strategies. A more recent approach uses a deep player behavior model [57] for a multiplayer online role game to generate an agent's behavior [56]. Those investigations aim to create Non-Player Characters, generalizing the people's behavior instead of just analyzing the best players.

The issue we face in this study is to find strategies in a database of matches from a videogame. Those strategies need to come from matches of players that achieved a high score in the videogame to give competitive solutions against experts in future works. The videogame we use to collect the matches is a model of the Housing Development Problem. In particular, our Housing Development model shares characteristics with the 2D Bin Packing Problem. Such similarities make us suppose that the strategy of Bottom-Left, a well-known heuristic for the 2D Bin Packing Problem, will result in successful matches.

To find strategies, first, we need to define the term for our videogame. We believe that the strategy of a player lies in the repeated sub-sequences of plays in a match. After knowing what we are looking for, we need computational tools capable of extracting such sub-sequences from the information of the videogame. The sub-sequences will help us tackle the problem of the study to describe strategies from the best players, and analyze the possibility of creating an algorithm that solves the Housing Development Problem using patterns as future work.

1.3 Objectives

The general objective of this work is to use the information extracted from a Videogame Crowdsourcing model for the Housing Development Problem, designed by ourselves, to obtain patterns that describe the strategies people use to get high scores. We aim to define a methodology to find useful information in matches from players and compare the plays that lead to different results. To achieve the final goal, the particular objectives of the research work are:

- Design a Videogame Crowdsourcing model capable of storing information about matches that solve instances of the Housing Development Problem.
- Use the information from the videogame to represent the plays of the matches collected.
- Extract patterns from matches to describe the strategies players use in the videogame to get high scores.
- Divide the matches by their score with a threshold, defined appealing to Pareto's principle. Such division should allow us to identify the differences between matches that can be taken to be winners from those that are not.

1.4 Hypothesis

Videogame Crowdsourcing models can obtain competitive solutions to problems from different people. From that information, we can get the process that leads a player to a high score. We believe that the strategy of a player lies in the repeated sub-sequences of their plays in a match. Obtaining those sub-sequences will help identify the strategies of the players in a videogame.

After formulating the hypothesis the research questions that arise for this project are:

- How can we model a Housing Development Problem as a Videogame Crowdsourcing?
- Can we interpret the information we get from a match to discover its strategy?
- What technique can we use to extract repeated sub-sequences from a match?
- Can we recognize strategies using the patterns obtained from matches?
- Do people follow strategies similar to the ones existing in the literature?
- Is there a difference between the strategies of people with high scores and the ones of people with low scores?

1.5 Solution Overview

In general, in the study, we explore the idea of finding strategies from a Videogame Crowdsourcing model; we focus on the Housing Development Problem. Several features describe this problem, making the solution space too wide. We abstracted the problem orienting the player to maximize the total number of houses, given the land for construction, subjected to certain restrictions.

For the solution, we propose a representation for a match of the videogame. Our model of the Housing Development Problem in the videogame shares characteristics with the 2D Bin Packing Problem. Those similarities allow us to anticipate that people can use heuristics from the 2D Bin Packing Problem in our videogame. Therefore, we oriented the representation to find strategies formed with plays similar to Bottom-Left, a well-known heuristic for the 2D Bin Packing Problem. The representation is in a sequence structure, allowing us to find the repeated sub-sequences or patterns we believe that form the strategy.

To get the patterns, we used Sequitur, an algorithm that extends the alphabet of a String with new grammar rules. It is capable of finding the repeated sub-sequences of the structure we defined for the matches. The experiment with this algorithm has two phases. The first one aims to validate whether our methodology is up to identify the strategy that we purposefully insert on each match; the second one obtains patterns from matches of players and identifies their strategies.

Sequitur used our methodology to identify five strategies from the best players. According to the results, if someone focuses on using one of those strategies, it will lead to a high score in the Housing Development Problem.

1.6 Main Contributions

This work studies the possibility of getting the strategy from matches of a videogame. The most important contributions are the analysis of the representation we designed and the strategies we found with it. The results showed that it is capable of abstracting details of the match while leaving the strategy in a sequence. This representation can work with videogames that follow a similar structure to the 2D Bin Packing Problem. With more details, the contributions are:

A Videogame Crowdsourcing model for the Housing Development Problem

The videogame used throughout the study is an abstraction for the Housing Development Problem we proposed. We focused it on the objective of maximizing the number of houses in the land to make a videogame easy for people to understand. This videogame provides a new model for the Housing Development Problem, which includes the houses and roads as available objects, and it can model other things by modifying the figure of the land.

A representation for matches of videogames similar to the 2D Bin Packing Problem

The methodology to translate an instance of the Housing Development Problem into our representation provides new knowledge to the literature. We focused such representation on describing a strategy of Bottom-Left, enabling it to explain a solution for a 2D Bin Packing Problem. Additionally, we proved that using this representation, we can obtain repeated sub-sequences, and more importantly, we can describe strategies with them.

Strategies that players use to solve the instance of the Housing Development Problem

With the patterns we extracted from matches of the videogame, we identified several strategies people use to get a high score; a score above the threshold we defined appealing to Pareto's principle. Among them, we confirmed that players use variants of Bottom-Left [27]. Regarding the other two strategies we detected in players, we could not find them as strategies for the Housing Development Problem or other problems with similarities to it, like the 2D Bin Packing Problem [9, 27, 37, 38, 40] or Architecture Layout Design Problems [3, 23, 42, 45].

1.7 Outline of the Thesis

The present document provides detailed information about the study previously described. Chapter 2 explains the concepts that this study covers, it includes: crowdsourcing and some models of Videogame Crowdsourcing, the task of pattern recognition, and similar problems to the one of videogame of this study. Chapter 3 describes the videogame we created and its method to evaluate the solutions that the user creates. Chapter 4 defines the representation designed and the algorithm used to get patterns in the study. Chapter 5 shows the phases of the experimentation using the matches from the database of the videogame. Finally, chapter 6 presents the conclusions of the study and the work that other investigations can take in the future.

Chapter 2

Related Background

The chapter presents an in-depth review of the literature concerning this study. It starts with the concept of crowdsourcing, showing the difference between tasks done by a group of people and this model. A platform needs to consider several elements to be a crowdsourcing model. Section 2.1 speaks about the essential components of crowdsourcing and details to consider to achieve its objective. Then, it mentions applications of Videogame Crowdsourcing, models that employ a videogame as motivation for people to participate, with some examples.

Another relevant concept for the study is player modeling, whose objective is to understand the behavior of players in a videogame to mimic their decisions. Among all the different methods to achieve that objective, the chapter focuses on the subject of this study, modeling a player's strategy. Some approaches to model a strategy consist of using a set of rules or machine learning algorithms to learn people's behavior from a database. Studies aim to predict the opponent's strategy or to build one trying to achieve specific goals. Among them, one research attempted to improve strategies by visualizing similar game states and the rules players applied in them. A closely related study used decision trees to create heuristics; however, it has some differences with our approach that section 2.2 mentions.

The chapter continues with some representations studies use to describe matches in different videogames for pattern mining. It mentions an investigation that studied data mining in a videogame for pattern extraction; it guided us to design the method to obtain information for identifying strategies. Then, it describes an application of the algorithm we used to find the repeated sub-sequences we defined as patterns. Finally, it reviews a problem that exists in the literature with similarities to our model of Videogame Crowdsourcing.

2.1 Crowdsourcing

"The new pool of cheap labor: everyday people using their spare cycles to create content, solve problems, even do corporate", that is how J. Howe described crowdsourcing in 2006 [24]. He was the person that coined the term for the first time in his article: The Rise of Crowdsourcing [24]. In the beginning, persons mistook crowdsourcing for any task done by a large group of people, like youtube. The truth is that crowdsourcing is a problem-solving model that needs an online platform. It uses people that get involved, also called crowd, in the model to find solutions.

Any crowdsourcing platform needs to focus not just on the task but on how to use the crowd. It takes advantage of the community's talents or labor to achieve an organizational goal. Therefore, processes in which companies get votes for a new product are not crowdsourcing because the control is mostly at the organization. They need to consider four components to apply the model into a platform [7]:

- A task that an organization needs to solve.
- A crowd that will work on the task voluntarily.
- An online platform to allow the interaction between the crowd and the organization.
- A benefit for the crowd and the organization.

As mentioned before, the model needs the crowd to work; it has to convince people to participate on the platform. Making the benefit or motivation an aspect the designers need to pay attention to before implementing the platform. Studies suggest that the individuals cooperate if one of the following motivations is involved [7]:

- Earning money.
- Develop creative skills.
- Network with other professionals or share something.
- Build a portfolio for future employment.
- Solve a challenging problem.
- Socialize and make friends.
- Do an activity when they are bored to have fun.
- Contribute to a common interest.

Depending on the problem it is facing, a crowdsourcing model can fall into a different category. Knowing the type of crowdsourcing can guide the design process and obtain better results. There exist four types of crowdsourcing [7]:

1. Knowledge discovery and management: The task of the crowd involves the search and collection of information. It is a method of the creation of collective resources.
2. Broadcast search: Assigns empirical or scientific problems to the crowd.
3. Peer-vetted creative production: Organizations ask people for creating and selecting creative ideas. Examples of these problems are design or aesthetic ones.
4. Distributed human intelligence tasking: The objective of the crowd is to analyze the data given to them.

In the literature, many studies have tried crowdsourcing in different areas. We can find examples in designing t-shirts, linking families, classify galaxies, and many others [69]. Nevertheless, we can categorize all of them into four subjects [7]:

- Design of a product.
- Find solutions to hard scientific problems.
- Reach an agreement on public issues.
- Process large amounts of data.

The last thing that these platforms need to know is how to handle intellectual property and copyright. The application must protect both the organization and the crowd. The most successful policies are easy to find and understand [7]. The challenges at Seeker companies make the individuals sign a legal agreement to protect confidential information and grant a temporary license to the possible decisions to submitted at the platform.

2.2 Videogame Crowdsourcing

Videogames are a solution to fulfill the requirements of a public platform and the motivation of the crowd. Each year, people spend millions of hours playing videogames [66]. The reasons for playing are many, either spend boring time, relieve stress, and many more. The purpose of this crowdsourcing approach is that people contribute to the problem without noticing they are doing some work.

Videogame Crowdsourcing has applications in different areas such as diverse security, computer vision, and internet search. The Peekaboom videogame, developed at Carnegie Mellon University [67], is a web videogame that has as an objective to help computers locate things in images. The videogame collects information about the picture, like the pixels that belong to an object; it obtained massive amounts of data from thousands of people that participated in the videogame [67]. A recent study provided insight into this computer vision task. The research gathered information about players on an image labeling videogame to compare them against descriptors assigned by an expert to the same images [26]. The study concludes that the tags provided by the users can serve as descriptors for pictures in different areas because of their high quality in comparison to the expert.

Another example is Phylo, a computer framework that represents DNA sequences by blocks of different colors [30]. Each color represents a nucleotide in the whole genetic code. The importance of the problem lies in the information it provides about the process of evolutionary patterns and the production of different species. The videogame simulates the multiple sequence alignment problem, which is NP-complete [68].

The model can be successful in collecting competitive solutions against experts. BioGames is another videogame that presents different red blood cell images to the crowd. Their task is to label them as healthy or infected [44]. This approach facilitates the process of training medical students to label diseases. It helps the platform to get valuable information about the problem while the crowd gets its reward by learning. The study proved that participants got an error of 2% compared to the result of experts on the same images. Related to a different aspect, the study remarks about the importance of assessing the legal issues before making the information available for the use of clinical settings.

Other studies combine the knowledge of people to improve the performance of machine learning models. EyeWire is a videogame that maps a three-dimensional reconstruction of neurons from the retina into images of two-dimensional slices [32]. The gameplay's objective is to follow the trace of a neuron. To do so, the player can color pixels near a neuron or in a new place where he/she thinks there is one. Since the videogame does not know the correct answer, it needs to compare different solutions to return a score. If a player colors spaces that many others did, he/she will obtain a high score. The purpose of this videogame is to aid a convolutional neural network trained to detect neural boundaries. This network encourages the players to focus their attention on places in which it is not sure of the correct answer.

The previous examples are interested in expanding the information on the solutions to a problem. In contrast, L. Perez et al. created a videogame for solving the robust facility location problem [17] to prove if players can outperform algorithms in the literature. This problem consists of a set of demand centers and potential locations for opening facilities. The objective is to find the places in which facilities will perform better. Also, the robust version requires a good response when the failure of a facility occurs. They relate this problem to the refugee aid deployment policy that consists of assisting and relocating refugee places. In the study, they compare the results of the players with algorithms and found that players have solutions that behaved better in some aspects.

An aspect that these videogames do not consider is cooperation among players. The study about the videogame Cell Evolution introduced a cooperative crowdsourcing model using blockchain technologies [72]. On it, players need to develop a single cell; however, the world of the videogame depends on millions of cells. For example, if a player decides to upload a toxic cell, it will affect the world negatively.

2.2.1 Player Modeling

Videogames are an industry that is quickly growing in popularity [33]. One reason is the diversity of the videogames and electronic devices. Besides, a sector, Electronic Sports (eSports), emerged to hire the most skilled players, and studies expect it to be a $23 billion industry by 2020 [33]. With this sector came a new interest in learning ways to play better by understanding the players' behavior, the form a person interacts under a videogame environment [48]. That information includes the strategy they used to solve the problem presented in the videogame.

Player modeling concern is to improve the abilities of a computer by generating algorithms capable of following players' behavior [4]. However, analyzing the data in a videogame is a difficult task. It needs a deep understanding of all the possible actions a player can make [48]. The problem is that there is a huge amount of data regarding player's decisions on a videogame. Those investigations aim to create Non-Player Characters, generalizing the people's behavior instead of just analyzing the best players. Some approaches use evolutionary algorithms [65], neural networks [50], deep player behavior models [56], and decision trees [2] to design algorithms that work as Non-Player Character.

The tools designed to analyze a player's behavior are useful to understand their strategy. Several kinds of research try to create models that enable them to study the interactions of a person within a videogame. Some applications visualize a player's navigation [14], use heat maps [16], define points of interest, where most actions happen, in a map [12], using one-hot encoding [31] or modeling an avatar's mobility with traces [10]. All of them aim to create a model for the player's behavior on a specific videogame.

There are several approaches to model a player's behavior; three of them are modeling actions, tactics, and strategies [4]. Modeling actions consist of game states combined with actions; it uses a likelihood function to select an action for every state. Modeling tactics use learning techniques to achieve local goals in a videogame. Finally, a model based on strategies uses tactic models and features to decide how to fulfill the final goal.

Strategy Modeling

In our research, we want to obtain the information from the player's behavior to model strategies. Regarding this subject, studies applied machine learning methods to build Non-Player Characters. There are attempts to design human-like agents at Super Mario Bros using neuroevolution [55], at racing games using evolutionary optimization [65], and neural networks [50]. However, few of them resulted in successful outcomes [73], like the one of Karpov et al. [29] that managed to pass the Turing test. In these studies, they created a strategy using the actions of a dataset of players' environment features and keys pressed during a match. To decide which action to use depending on the situation, they use machine learning algorithms.

Although those machine learning algorithms can represent a strategy [55, 65, 50, 29], their models focus on predicting an action for the current state instead of thinking about the whole plan. A model that focuses on finding actions consist of a set of rules: $R = \{(s_0, a_0, e_0), (s_1, a_1, e_1), ..., (s_n, a_n, e_n)\}$ where s_i is a state of the videogame, a_i is the action to make in that state, and e_i is the effect that occurs to the current state by performing a_i; it is important to notice that e_i might not be explicit in these models, but every action a_i they take at state s_i produces the new state e_i. At this set of rules, two triplets can be the same, and a rule may not produce a new state existing at another rule from the set. The first statement indicates that in the set of rules it can exist two triplets such that $(s_i, a_i, e_i) = (s_j, a_j, e_j)$, for $i \neq j$. The second statement says that a rule r in the set R can have an effect e_i such that no rule element of R has a state $s = e_i$; in that case, the algorithm uses a method to find the closest state of a rule to the unknown one and make its action.

On the other hand, a strategy that focuses on the whole plan does not have the two statements of the models that predict actions. The set of rules for this type of method consists of actions and states $R = \{(s_0, a_0), (s_1, a_1), ..., (s_n, a_n)\}$. The difference is that the actions of this model do not produce an unknown state, and the set does not have repeated rules. Therefore, every action a of a rule in R produces a state s of one rule r in R; also there is no pair $(s_i, a_i) = (s_j, a_j)$, for $i \neq j$ and i, j taking values from 1 to n. Nevertheless, this second approach might not be viable for every videogame; for multiagent videogames, the player has no full control. For example, in Super Mario Bros [55], the objects and the map are not always the same, and the player cannot modify those variables, then it is necessary to decide the action in real-time. Our videogame is a puzzle-type videogame; hence the user is the only one controlling the states of the videogame, allowing us to search for whole plan models.

A group of videogames that attracted a lot of research is Real-time strategy games. In them, players control a set of units distributed on a map, and actions occur in real-time, meaning that they cannot store information during turns [20]. Existing studies apply Bayesian approaches to predict initial strategies [60], evolutionary algorithms to define the behavior according to a set of rules [19], and Case-Injected Genetic algorithm to generate strategies [39]. In contrast with the studies mentioned previously, here they define a strategy as a set of rules; they establish them based on experience or a structure that guides the videogame. There are two options that the studies use the predicted strategy in these games, either to detect the opponent's strategy or to complete specific goals. In a videogame called Starcraft, Weber, and Mateas [70] apply machine learning approaches to predict the opponent's strategy using an existing database of experts. Another study by Ontañon et al. [53] builds a framework for Wargus, a videogame. To train their system, they use traces from experts that include the actions made to complete specific goals. At execution time, it creates a plan tree that consists of goals it wants to fulfill and the actions needed to do it.

A research that is closely related to our works in a robot soccer game [59]. They define a strategy as a set of rules for specific descriptions of the objects on the map. Each rule is a quaternion of the coordinates of the player's robots, the coordinates of the opponent's robots, the coordinates of the ball, and the coordinates of where the player's robots should move. At every step, the algorithm compares the current game field situation with the strategy's rules to select the closest one. However, a rule in the strategy may not connect with others; there might also be duplicate ones. Using a dataset of matches, they extract sequences of game situations, and with a clustering algorithm, they visualize the relations between them. Knowing the different game situations, they believe it is possible to improve strategies for robot soccer games. The main difference between our study and this one is that their model bases on actions and not on the whole plan.

A more recent approach uses a videogame of a 2D Bin Packing Problem to get human players [58]. The study got ten players and build decision trees using their matches to create strategies. This approach uses a whole plan method, but it differs from the one we propose. First, they do not make a distinction between the solution of the players, meaning that they can make a heuristic using a solution that performed poorly in comparison with others. The second reason is that their plan concentrates on packing one item; instead, ours focuses on filling the entire land for a Housing Development Problem.

2.2.2 Videogame Representation

The representations used for videogames we mentioned previously relate to the one we need for our videogame. However, the closest one is from the research of Cavendeti et al [12] because it works in a single map, analyzing key actions of players in it. The study focuses on a videogame called Defense Of The Ancients 2 (DOTA2), an eSport strategy videogame that contains traces and choices. Their objective is to find paths that can help a player understand his/her decisions during the match and improve his/her gameplay. To complete it, they needed a method to translate the data from a replay of a player into features and obtain patterns. The information each replay included is:

- The coordinates (x, y) on the map of the videogame.
- The upgrades applied to the character of a player.

- The items and abilities a player bought during the match.
- The gold and experience the player win over time.
- Global information about the match. An example can be the winning team.

The representation to obtain patterns of the plays consists of four steps [12]. The one relevant to our research is the Contextualized Trajectory Generation. It translates the coordinates of the map into points of interest (POI), regions in which most of the plays happened. With such division, Cavendeti et al [12] form a sequence with all the POI, eliminating the ones repeated consecutively, with the actions that took place there. Our representation of the Videogame Crowdsourcing's solutions is related to this step since it is composed of a sequence of actions that occurred in specific spaces of our map.

This model can mine frequent patterns on positive plays, according to discriminant pattern mining. They achieved a methodology that allows to output patterns that can explain the reasons a strategy leads to victory and the differences between a reference behavior from the best players.

A more recent study employs an algorithm called TRACE [11] to represent a match of a Multiplayer Online Battle Arena videogame with a trace of avatar movements [10]. The objective of the research is to provide insight into avatar movements for studies that aim to develop mobility models for this type of videogames.

2.3 Pattern Recognition

As mentioned in the previous section, a strategy is composed of a group of actions. In a videogame, those actions are plays the user made. One way to define them is by using patterns. A pattern or descriptor can be expressed as conditionals that evaluate true or false [8]. As an example, a pattern can describe a match where the user played action 1 ($A1$) after action 4 ($A4$); it will be $A4 \rightarrow A1 = True$.

The task of pattern mining consists of an approach to discover relevant patterns from a database [5]; several methods attempt to get solutions for this problem [8, 51, 64]. Depending on the task, some of them can perform better than the others. For our information, extracted from the Videogame Crowdsourcing, we were searching for a miner capable of getting patterns from a sequence of characters. Such sequence is a structure we define to represent a match from the videogame.

To find patterns with our structure of a match, we performed tests with three different miners, PBC4cip [41], Apriori [61], and Sequitur[52]; in each case, we adapted the sequence to the especial needs for each algorithm. To test it with PBC4cip, we defined variables for all the possible sub-sequences of size one and two. The problem with this approach is that the sub-sequence space increases exponentially as we add variables of sub-sequences with different sizes. For instance, if we wanted to add sub-sequences with a length of three, the amount of variables we need to add is higher than 9000. Additionally, most of those features have a value of zero for a sequence, adding noise to the dataset. Whereas, when working with Apriori, we found associative patterns between two elements of the sequences. Lastly, Sequitur defines rules that represent the repeated sub-sequences in a match using our sequence structure. After these tests, we found that Sequitur obtained better results than the other miners we tried. The reason is that we defined a match as a sequence of actions, where the patterns are repeated sub-sequences in it, making Sequitur a more efficient algorithm for this task.

2.3.1 Detection with Customized Grammar

Sequitur, as we explained previously, is the algorithm we found to be more useful to get the repeated sub-sequences we needed from the Videogame Crowdsourcing. Its objective is to extract hierarchical structures from a text where the context does not matter [52]. It analyzes a sequence of characters and builds grammatical rules in linear time. Taking into consideration the repetition of the patterns, Sequitur creates new rules; a sub-sequence has to appear at least twice in the string to form one.

The study of Latendresse [34] used this algorithm for masquerade detection, a cyber attack in which someone impersonates a user of a system. The objective is to detect anomalies in the commands and find the intruders. The first step of the algorithm is to build user profiles. With uncontaminated users, Sequitur generates grammar rules from sequences of commands. Each grammar rule finds the total and global frequency (f_p and F_p) on all the profiles. Here, f_p is the frequency of a command from a user, while F_p is the frequency among all users.

To detect a masquerade, the study [34] uses another dataset that contains sequences of commands that might be contaminated. With an evaluation function and a global threshold, the study predicts if there is an intruder. The first step is to divide sequences into blocks; then, calculate the value of the evaluation function for each block. At last, compare the value with the global threshold; if it is at least the same as the threshold, the block is considered normal.

2.4 2D Bin Packing Problem

In this study, we design a Videogame Crowdsourcing model for the Housing Development Problem. After some research, we found the two dimensional (2D) Bin Packing Problem shares some similarities with our model. We took advantage of those similarities to define our representation of a solution and the validation case using existing strategies for this problem that prove our method works correctly. However, dividing the Housing Development Problem into sub-problems of the 2D Bin Packing Problem does not guarantee a valid solution. The reason is that the items at the 2D Bin Packing Problem do not share any relation. Therefore, joining sub-problems of the different types of objects from a Housing Development Problem can provoke an invalid solution because one item does not meet a restriction. In this section, we describe the problem with the features that explain an instance and the solution space. We provide some of the solution techniques, including the one we used to define our method, Bottom-Left.

2.4.1 Description

There exist different types of Bin Packing Problems. Each of them has its characteristics, but all of them belong to the NP-Hard problems [62]. The classical 2D Bin Packing Problems consist of a set of rectangular items that need to be packed into 2D bins with fixed width and variable-length [6]. A variant of the problem works not only with rectangles but with irregular shapes too (2DIBPP) [1]. A recent study tackles this problem with an approach using videogames to train decision trees and obtain Human-Derived-Heuristics [58].

For a formal description of the 2D Irregular Bin Packing Problem [1], let P be a set of polygons $\{p_1, ..., p_n\}$ that needs to be packed into rectangular bins. The bins have a fixed width W and length L. A possible solution is described by a set $B = \{b_1, ..., b_N\}$, where N is the number of bins required to pack n polygons. The packed items placed on bin b_j are described as $P_j \subseteq P$ and the number of items in b_j is denoted as n_j. All packed items p_i have an orientation angle $o_i \in [0, 360)$ degrees and their location on the bin. The location is described by (x_i, y_i) coordinates, taking the origin as the bottom left corner of the bin.

The objective of the problem is to minimize N following three constraints [6]:

1. All of the items must not overlap with each other.
2. Each item can be packed only once.
3. The items placed on a bin must not exceed the dimensions of it.

2.4.2 Instance Features

To differentiate instances of 2D Bin Packing, we use features to describe them. In the study of Mexicano A. et al. [46], they tested 27 features obtained from the literature to find the metrics that best describe this problem. Their results showed that 8 of them were relevant for classifying an instance. At them, n is the number of items, and w_i is the size of item i; they are the following:

Size of the instance (s): Relationship between the size of the instance and the maximum size that has been solved.

$$s = \frac{n}{n_{max}} \tag{2.1}$$

Factors: The proportion of items with sizes that are a factor of the bin capacity. Item i will be a factor if the bin has a capacity that is multiple of w_i.

Use of bins: This feature uses the sum of all the sizes of the items (Sw) and the bin capacity (c). It quantifies the proportion of the items that can be packed in one bin. If the $c > Sw$, the result of this feature is 1. Otherwise, the result is c/Sw.

Mean: The average size of the items.

Range: Difference between the maximum and minimum values of the sizes of all the items with respect to the bin size c.

Asymmetry Pearson Coefficient (A_p): Comparison between the mean and all the sizes of the items at an instance.

$$A_p = \frac{\sum_{i=1}^{n}(w_i - avg)^3}{n\sigma^3} \tag{2.2}$$

Minimum: Shows the minimum size with respect to the bin size c.

Repetition Frequency: Calculates the frequency of repetition for each size that the items have and it shows the size with maximum frequency.

2.4.3 Heuristics

We can divide the solution for 2D Bin Packing into two parts: the first one is the assignment of the items to the bins; the second is the placement of the item into the bin [1]. Heuristics are methods that estimate the cost of going from an initial state to the goal state. Some placement heuristics designed by researchers are:

Bottom-Left (BL): It begins by putting the first item at the bottom left corner. The next item starts at the upper right corner; then, it is slid to the bottom until it hits another item and then to the left until it hits an item. When the item reaches a stable position, it takes it. This heuristic stops when there are no more items or space in the bin [27].

Improved Bottom-Left (BLLT): Similar to the BL heuristic. The difference is that it prioritizes the bottom movement over the left movement. If the current item can move to the bottom, it will continue until it hits an item. Then it will move to the left and check if the bottom movement is available again [37].

Minimum Length (ML): The heuristic selects the position that minimizes a distance. This distance is between the origin of the bin and the right-most x coordinate of the item [37].

An example of an assignment heuristic is:

Firs Fit Decreasing (FFD): The heuristic orders the available bins by non-decreasing size, and when a new item arrives, it selects the first bin in which it fits to pack it [15].

A recent method to solve this problem uses FFD to assign an item into an available bin, then it uses BL and an exchange method to pack the item on the bin, and it performs a final local search to improve the solution [38].

2.5 Summary

In this chapter, we introduce the model of crowdsourcing with the essential components for it to work. The model does not focus on tasks done by a large group; it needs to have a problem that some organization needs to solve. For our case, we want to solve the Housing Development Problem, fulfilling that requirement of the model. Another relevant concept we mention is the motivation for people to participate. Among all the different ones, we work with Videogame Crowdsourcing. All the examples we discuss here try to solve problems or build databases. An interesting fact they found is that the results of players can compete against expert's solutions.

Player modeling is a research area oriented to the goal of this study. It analyzes the behavior of people in videogames to create algorithms that solve them. The objective of the investigations we mention in the chapter is to create Non-Player Characters to improve the experience of people. They are closely related to our work since some of them try to model strategies employing different approaches. In general, to find a strategy, they use a set of rules and a machine-learning algorithm to learn when to use them. Nevertheless, some of the rules within a strategy might not be related or can be very similar to others, as we explained for methods that aim to predict actions.

To find the repeated sub-sequences or patterns from our videogame, we need to find a way to represent a match. Some of the approaches we mention here use a method that divides the map of the videogame into regions. Since our videogame consists of a fixed area, it is viable to use that method to define our matches. Another relevant aspect of our videogame is the sequence of plays. Sequitur is an algorithm that can help us get repeated sub-sequences without altering the order of the original. Here, we saw an application of it in a solution approach to anomaly detection via grammar rules.

The study focuses on the problem of Housing Development. To create a Videogame Crowdsourcing, we need a model that most people will be able to understand. The 2D Bin Packing, which we describe in this chapter, is a well-known problem that shares similarities with our model. The objectives are similar, but Housing Development has some additional restrictions because it involves different objects that have a relation between them. An advantage of using the 2D Bin Packing of these similarities is that we can utilize some of the existing heuristics that solve this problem to validate the methodology we design to extract strategies.

Chapter 3

Videogame Design

The videogame is an abstraction of the Housing Development Problem. We chose the problem because it is of interest to the architecture community. There is a low supply of houses in places where the urbanization rate is high. One reason for that is the few residential spaces available for construction [63]. Architects need to optimize the available spaces to give as many houses as possible to cover the demand. Providing solutions to it can simplify the work of architects. Additionally, this problem is similar to another that is of scientific and industrial interest, the 2D Bin Packing Problem. This chapter describes the details behind the implementation. Section 3.1 mentions the reasons behind the design. Section 3.2 describes how the videogame evaluates all the conditions needed to obtain the strategies we desire. Sections 3.3 and 3.4 show the interface and the information on the database.

3.1 Videogame Design

A Housing Development Problem has many variables making it hard for all people to understand it. Architectural Layout Design Problems, such as the Housing Development Problem, consider features of cost, duration, aesthetics, among others; each of them involves a variable of the videogame. A player needs to know all the variables affecting a solution to perform well in his/her match. Players might drop a videogame if it gets too complicated; having more variables makes a problem harder to understand. Our objective is to get as many players as possible. Then, we need to simplify the problem, focusing on what we want to optimize. We abstract subjective features, like aesthetics, and focus the videogame on optimizing the space for houses, maintaining the reason for choosing the Housing Development Problem.

Our abstraction of the Housing Development Problem focuses on maximizing the total number of houses, given the land for construction, subject to certain restrictions. The videogame model is similar to a 2D Bin Packing Problem; we can consider the items as the houses and the bin as the land. It challenges the player, similar to the 2D Bin Packing Problem, to put as many houses as possible on the land. The difference relies on connectivity restrictions. Our model on the videogame forces the players to maintain all the houses connected, making it different from a 2D Bin Packing Problem.

A residential complex needs connectivity between its houses; it must have one entrance that links the houses with the exterior. The entry allows people to get to their homes either by car or walking; assuming that there is just one, all the houses are connected. The videogame uses this assumption to force the player to link all the houses between them. A player can use an item that represents a road to meet the restrictions of connectivity. The videogame, however, does not model the entrance; it is easier for an architect to find a place for the entry with all the houses connected.

The figure for the land is a relevant variable in the videogame. A limitation of architectural problems like Housing Development is the available area [13]; the initial shape of the land depends on the project. Another factor that affects the area available for houses consists of the different objects that the architect includes in the residential complex; they can be trees, lakes, parks, among others. Changing the shape of the land, at the videogame, can abstract other objects. This research, nevertheless, considers just one figure for the land. Evaluating the strategies on different land shapes went beyond the scope of the study.

The input for the Housing Development Problem is defined by an area for the land a_l, at least one area for a model of a house a_h, and the area for a road a_r. The objective of the problem is to maximize the amount of houses x in the land. Equation 3.1 represents the goal of the Housing Development Problem without the restriction of connectivity, where y is the number of roads. The restriction is described in section 3.2.4. The output of the problem is composed of the positions of the roads and houses in the land.

$$max_x : a_l - xa_h - ya_r \geq 0 \tag{3.1}$$

The implementation of the videogame considered two points to select the tools. The first one is the ease of distribution among people, which we resolved using a web page, a media capable of getting into a large group of people. To create it, we used Phaser2, a framework of Javascript that has functions for videogame design. The second point is the data capture in real-time; for this, we need a platform with the ability to store information. Javascript can use Cloud Firestore, a NoSQL database from Firestore that works in real-time. It receives the information on every play a player performs. The tools we selected meet the requirements of the two points we need for the Videogame Crowdsourcing method.

The method we designed is a broadcast search crowdsourcing because we are assigning an empirical problem to the crowd. We call the method Videogame Crowdsourcing because the platform is a videogame, and it counts with the four components of crowdsourcing applications [7]:

- Task to solve: find strategies for the Housing Development Problem.
- Crowd: people on the internet.
- Online platform: the videogame presented in this chapter.
- Benefit: entertainment.

3.2 Videogame Description

The initial state of the videogame presents the player with a fixed polygon and another one that can change among four options. The fixed polygon represents the available $land$ to place houses; the other one represents the items the player can place. The choices the player has for the items are three different polygons for the $houses$ and a rectangular figure for the $roads$. The player can choose to place any of these four items at any time during his/her match.

The objective of the videogame is to place as many none overlapping $houses$ as possible, maintaining the connection between them with a path composed of $roads$. The number of $houses$ in the solution gives the information to calculate the score; it encourages the player to cover as much space he/she can with $houses$. The answer is the final set of $roads$ and $houses$ in the $land$. To meet the restriction of connectivity, the player includes $roads$ on his/her solution. A $road$ must touch a $house$ to form a link; all $roads$ linked between them form a path. A complete solution has one path and a set of houses.

An object, either the land or an item, at the videogame consists of a center point (x_c, y_c) and a set of points P. The set P, in equation 3.2, represents the n vertices of the polygon.

$$P = [(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)] \tag{3.2}$$

We designed set P to have the edges of a vertex adjacent to it; vertex p_i forms edges with p_{i-1} and p_{i+1}. The ordered pair of points from P defines the set of edges E. The videogame represents set E as equation 3.3.

$$E = \{(x_i, y_i), (x_{i+1}, y_{i+1}) | 1 \leq i \leq n-1\} \cup \{(x_n, y_n), (x_1, y_1)\} \tag{3.3}$$

The action of placing an item on the $land$ is called a play; each play adds one item to the solution. The player can select any item in the solution to delete, change, or move it.

3.2.1 Valid House Play

A $house$ play is evaluated by the videogame to let the player know if it does not violate any condition. A valid $house$ play needs to be inside the polygon of the $land$, and the $house$ cannot collide with another. If the $house$ does not meet one condition, the videogame will not allow the player to continue. This section describes how the videogame validates both conditions.

Condition: Contained by Land

The player can perform a $house$ play by putting the $house$ inside the $land$. To confirm the placement the videogame uses the Contains Algorithm [74]. This algorithm checks if the $land$ contains an item O with vertices P_o. The input $E_{Object1}$ of the algorithm is the set of edges of $land$, E_{land}; $P_{Object2}$ is a point $p_o \in P_o$. The algorithm calculates how many edges of E_{land} the point p_o intersects.

The Contains Algorithm evaluates three scenarios to know if a point intersects an edge. It begins selecting an $edge \in E_{land}$, formed by two points (e_1, e_2). To evaluate the scenarios, it forces two conditions:

- e_1 needs to have the smallest value on the y axis. Otherwise, change e_1 with e_2.
- p_o cannot have the same value on the y axis as e_1 or e_2. Otherwise, add a small constant to the y value of p_o

There are three scenarios, in figure 3.2.1, for a plane with one point and one edge:

- The first has p_o above, below or right of $edge$. Contains Algorithm uses x and y verify the scenario.
- The second one checks if the p_o is left of $edge$. Contains Algorithm uses x value to verify the scenario.
- The last one is the case where one point of $edge$ is to the left of p_o and the other right. This scenario is the only one that can be positive or negative. Contains Algorithm calculates the slope m_1, between e_1, and p_o, and the slope m_2, between e_1 and e_2. It evaluates the cases that the other two scenarios cannot, using both slopes.

The second scenario confirms an intersection of p_o with $edge$; the first scenario denies an intersection; the third scenario has two cases, one with an intersection and the other with none. The algorithm evaluates one at a time and finds one that fits the input. We use this algorithm to describe the Contain operation of equation 3.4.

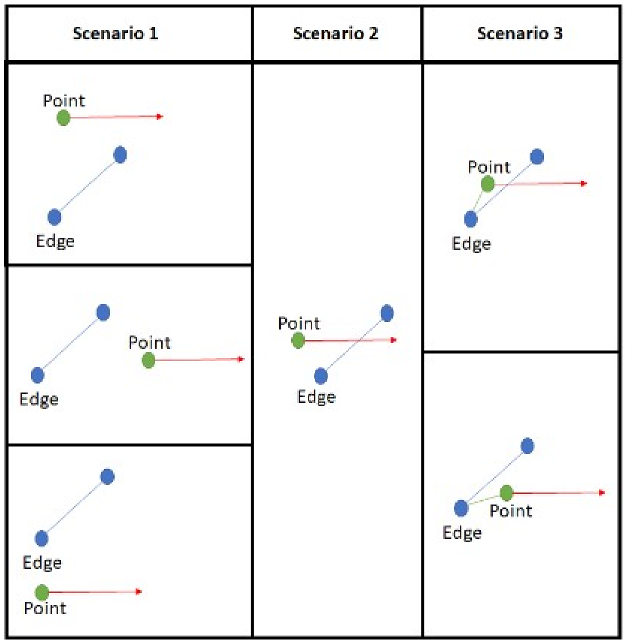


Figure 3.1: Scenarios that the Contains Algorithm searches.

$$\begin{array}{ll} Object.Contains(Point) = True & if\ intersections\ is\ odd \\ Object.Contains(Point) = False & otherwise \end{array} \tag{3.4}$$

If there is no vertex of the $house$ outside the $land$, we confirm that the $land$ contains the $house$. Equation 3.4 applies the Contains Algorithm; it confirms or denies if the point is inside a polygon. It tells us, applying it to all the vertices of a $house$ if it is inside the $land$. Equation 3.5 is the first condition to validate a $house$ play.

$$\begin{aligned} &land.Contains(house) = True && \forall p \in house, land.Contains(p) = True \\ &land.Contains(house) = False && otherwise \end{aligned} \tag{3.5}$$

Algorithm 1 Contains Algorithm

```
input  : $E_{Object1}$: An array with the edges $[(x_i, y_i), (x_j, y_j)]$ of a polygon
input  : $p_{Object2}$: A point $(x_k, y_k)$.
output: intersections: the times $P_{Object2}$ intersects an edge in $E_{Object1}$
$intersections = 0$
for $edge\ in\ E_{Object1}$ do
    $e_2 = edge[0]$
    $e_1 = edge[1]$
    if $e_1.y > e_2.y$ then
        Change $e_1$ with $e_2$
    if $y_k == e_1.y\ or\ y_k == e_2.y$ then
        Add a small constant to $y_k$
    if $y_k < e_1.y\ or\ y_k > e_2.y\ or\ x_k > max(e_1.x, e_2.x)$ then
        continue
    if $x_k < min(e_1.x, e_2.x)$ then
        intersections +=1
        continue
    else
        $m_1$ = calculate slope between $e_1$ and $P_{Object2}$
        $m_2$ = calculate slope between $e_1$ and $e_2$
        if $m_1 \geq m_2$ then
            intersections +=1
return intersections
```

Condition: No Collisions

A collision between two *houses* happens when one vertex of one *house* is inside another. The second condition for a valid *house* play does not allow collisions of *houses*. The same process to check intersections can find if there is a collision between two items. When a *house* collides with another, equation 3.4 will be true for at least one vertex of the *house*.

To evaluate the condition the videogame uses the set in equation 3.6, of k *houses* in the current solution. It starts using a *house* $h_i \in H$ and set of equation 3.7, $H_{-i} \subseteq H$. The information allows the videogame to check if the h_i collides with another *house* of the current solution. The Collide operation in equation 3.8 uses the Contain operation to find the existence of a collision between two houses.

$$H = \{house : land.Contains(house) = True\} \tag{3.6}$$

$$H_{-i} = \{h_1, h_2, ..., h_{i-1}, h_{i+1}, ..., h_k\} \tag{3.7}$$

$$\begin{array}{ll} h.Collides(h_i) = False & if\ (\forall p \in P_{h_i}, h.Contains(p) = False) \\ h.Collides(h_i) = True & otherwise \end{array} \tag{3.8}$$

The videogame needs to ensure that there is not a single collision in the solution. Equation 3.8 works for two houses; the videogame compares h_i with the whole set H_{-i}. The operation needs to be done with $h_i \in H$ for $1 \leq i \leq k$. Each cycle evaluates if a $house$ play is not valid due to a collision. If a $house$ play passes this condition and the Contain condition, it is valid.

3.2.2 Valid Road Play

A $road$ play needs to pass two conditions. The videogame evaluates the $roads$ in the current solution, from equation 3.9. The first condition is the Contain condition, described in section 3.2.1. The second one does not allow a $road$ to be inside a house. If a $road$ does not meet one of them, the play is not valid.

$$R = \{road : land.Contains(road) = True\} \tag{3.9}$$

There are several ways to evaluate if a $house$ contains a $road$. To validate the connectivity restriction, the $house$ needs to collide with a $road$. The videogame avoids the action of players putting $roads$ inside a $house$, allowing collisions, considering a point that is not a vertex of the $road$. The point that cannot be inside a $house$ is the center point of a $road$, $r_c = (x_c, y_c)$. The videogame uses equation 3.10 to validate the play.

$$\forall r \in R_p\{\nexists h \in H : h.Contains(r_c) = True\} \tag{3.10}$$

3.2.3 Operations

The videogame has two operations to modify the position of an item, either a $road$ or a $house$ before placing it; they are translation and rotation. They allow the player to choose the most convenient position for an item.

Translation The position of an object depends on its center point, (x_{c1}, y_{c1}). When the player selects a new position for the item, the center will change into the point, (x_{c2}, y_{c2}), selected by the player. Sets P and E are modified accordingly. The angle a that defines the orientation of the item does not change with this operation.

Rotation This operation can produce a rotation of the item from 0 to 360 degrees. The angle a is the one that defines the position of the vertices in P. One of the two lines that define a is the one produced by the center point of the item and point $p_i \in P$; the second is the horizontal line. The player can increase or decrease the value of the angle to rotate the item.

3.2.4 End of the Videogame

The user can select to evaluate his/her solution at any time of the match. The evaluation gives a score to the match if it meets all the restrictions. After the user gets a score, he/she can finish the match. A solution must be a complete one to obtain a score. Previous sections mentioned the restrictions of connectivity. For that, all the $houses$ must maintain a connection between them with a path composed of $roads$. Such path consists of a set of $roads$ in which there is no $road$ disconnected; a collision between two $roads$ connects the paths that belong to them. A solution that has at least one $house$ and connects all the $houses$ with a single path is complete.

The videogame performs two operations to check if there is only one path at the solution. It finds the collisions of each $road$, from the set of equation 3.9, and it uses the Check Path algorithm. Equation 3.8 can compute the collisions of each $road$. The $roads$ that collide with a $road$, $r_i \in R$, form a set $r_{i_{set}}$, for $1 \leq i \leq k$ and k being the length of R. The operation returns k sets, each one containing the collisions of one $road$.

The algorithm Check Path, a depth-first search method, needs the k sets of collisions, obtained on the previous operation. The algorithm starts with the first $road$ play, r_1, and inserts it into a new empty set, called $visited$. Then, it takes one $road$ from $r_{1_{set}}$, and repeats the process with this new $road$. If the road it takes belongs to $visited$, the recursion breaks at that point. The algorithm stops when the set of collisions of the $road$ it is checking is exhausted. All the $roads$ that form a single path belong to $visited$. If the sets R and $visited$ are the same, then all the $roads$ form a single path.

Algorithm 2 Check Path

```
input : $r$: A road in the solution
input : $r_{set}$: An array with all the roads that collision with $r$
Initialize empty set $visited$
Add $r$ to $visited$
for $road$ in $r_{set}$ do
    if $road \notin visited$ then
        Add $CheckPath(road, road_{set})$ to $visited$
return $visited$
```

The videogame needs to verify if the path in the solution connects all the $houses$. That test validates the restriction of connectivity. The Collision operation can confirm that a $house$ connects to a $road$ or the other way around. Equation 3.11 describes how to validate that the solution is complete, given that there is just one path. If the solution is complete, the user can end the match.

$$\forall h \in H\{\exists r \in path : h.Collides(r) = True \vee r.Collides(h) = True\} \tag{3.11}$$

3.2.5 Getting a Score

The videogame can evaluate a solution at any time of a match. The videogame will not allow the player to make an invalid play, according to the conditions of sections 3.2.1 and 3.2.2;

the solution contains only valid $house$ and $road$ plays. A valid solution can be complete or incomplete, according to section 3.2.4. If it is incomplete the videogame gives a null score, and the player cannot finish the match.

When a solution is complete, the score that the videogame gives is k, the length of the set H at equation 3.6. The objective we want the player to focus on is maximizing the number of $houses$. If the videogame gave extra points for placing different figures, the number of $roads$, the relation between items, etc., the player will get distracted on other objectives. Therefore, it only counts the points from placing a house, ensuring that the strategies players use are the ones we need.

3.3 Interface of the Videogame

The videogame is on a web page available to any person with access to the internet (https://arturosg3.github.io/Bin-Packing-Videogame/index.html). The languages used in the code are JavaScript, HTML, and CSS; the deployment to a web page in GitHub is possible due to them. To help with the interface, we use a framework of JavaScript called Phaser2; Phaser2 has commands to create videogames. The result of the program is the interface presented in figure 3.3.

The available $land$, to place the solution, is presented as a green polygon. The videogame challenges the player to place as many $houses$ as possible, given the available $house$ models. The player can see all the available models during his/her match. The item that the player is going to place has a pink color; it can change into a different $house$ model or $road$. All the items that form part of the solution are in the $land$.

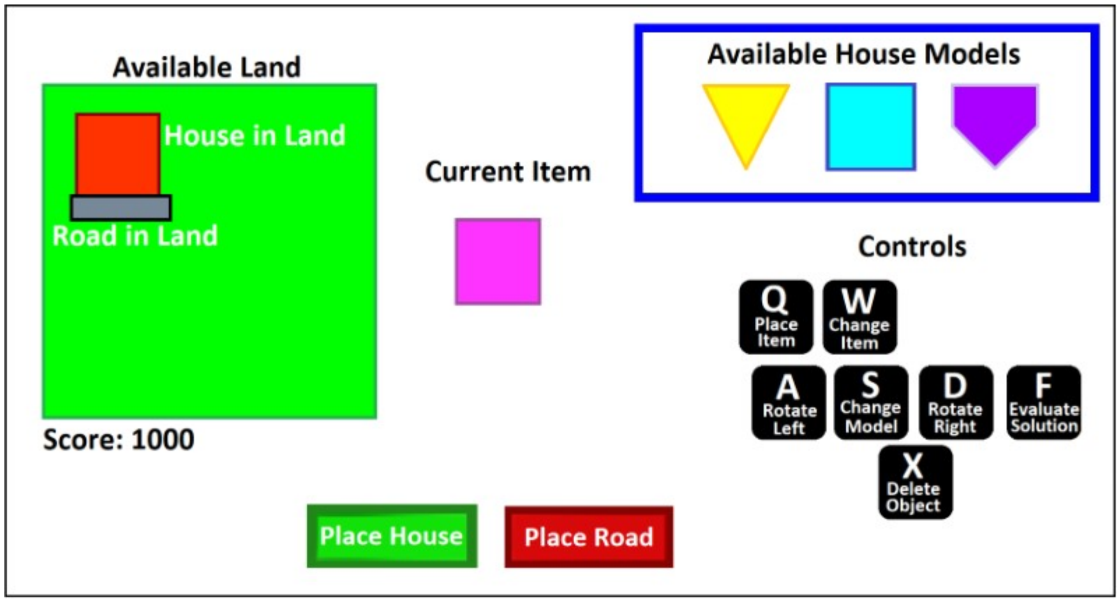


Figure 3.2: Interface of the videogame.

3.4 Database

A player follows several steps to construct a solution; a database stores some of them that can help us to reconstruct the procedure of the match. Cloud Firestore is a NoSQL platform from Firebase which can store data from a web page. It connects with the interface of section 3.3. The advantage of this service is that it synchronizes the information as the player updates it. The database contains information on all the decisions that each player made during a match. It consists of three parts:

- The parameters of the videogame.
- The description of any solution.
- The procedure of the match.

The parameters of the videogame consist of the information extracted from the $land$. This information is:

Vertices of *land*: as described in the previous section, one of the parameters of an object at the videogame is a set of points P. It contains the vertices of the polygon. The length of P is the parameter extracted as the number of vertices.

Area of *land*: knowing the polygon of the $land$, its area is calculated accordingly.

The description of the solution contains information about all the items inside the $land$. The solutions can be complete or incomplete. However, all the incomplete solutions are stored with a final score of 0. This information is updated when a player finishes the videogame. It includes:

Amount of *houses*: the total amount of $houses$ placed on the $land$.

Amount of *roads*: the total amount of $roads$ placed on the $land$.

Free area of *land*: the area of the $land$ that is not occupied by $houses$.

Score: The final score achieved by the player.

The procedure of the match contains the individual information of all the items. If the player places or eliminates an item, the videogame stores the data. It contains:

Placed order: the order in which the player placed all the items.

Elimination order: the order and time of deletion if the player decided to delete one or more items during the play.

Item's type: the type of an item placed or eliminated. It can be either one of the three models of $houses$ or a $road$.

Item's Position: the center point (x_c, y_c) for all the items placed or eliminated.

Item's Angle: The angle of an item is either placed or eliminated.

3.5 Summary

This chapter presented the design and implementation of a Videogame Crowdsourcing method. We abstracted the Housing Development Problem and designed a videogame web interface. The objective of the videogame is to place as many $houses$ as possible. The videogame focus on validating the plays and confirming the solutions meet the restrictions of connectivity. All the process of the match contains the strategy of a player. The information of the process goes to a real-time database. Chapters 4 and 5 explain how to use this information to obtain the strategies of the players.

Chapter 4

Sequence Representation for Strategy Mining

Our videogame has been designed to store as much information as required to replay every match held by a player. Mining a player's strategy from a given videogame match is not straightforward because the strategy is reified throughout all the plays of that match. The problem is further magnified if, associated with each player, we have several matches, and there is no certainty the player was faithful to his/her strategy at all times. Furthermore, the action space of our videogame is huge, for there are too many options for a play, making it hard to discover a strategy using our videogame representation. That is the reason why we have abstracted out much of the detail from a match, to reduce the play's search space.

To begin with, in our videogame, a player may place a house or a road anywhere in the given piece of land. However, in our abstract representation of a match, either of these objects can be placed only in a few designated areas. Each of these areas is a brick in a 2D regular grid, which is big enough to contain both a house and a piece of road that guarantees that house's connectivity to the rest of the neighborhood. More specifically, we can consider a brick as a unit that meets the requirements of the videogame. To transform a match from the videogame into our representation we can use all the house plays, knowing they will lie in different bricks.

The new representation provides a transformation for the solution of a match. It assumes that all the house plays have the same characteristics, except for the location. Additionally, instead of expressing the position with real numbers, the abstract representation uses the coordinates of the brick in the grid where a house lies. The transformed solution consists of the ordered sequence of the nonempty bricks' coordinates.

This abstraction allows us to translate a match, conformed by all the plays, into a sequence with fewer details. Then, we make another abstraction to reduce the play's space even further; it consists of transforming it into a sequence of movements. Each movement involves placing a house on a brick followed by displacement to another brick in which the player has put the next house. We define four different kinds: horizontal, vertical, diagonal, and special movements. The first three describe adjacent movements, while the fourth one describes the ones that end in a border. Since our videogame has similarities with the 2D Bin Packing problem, we decided to make the representation capable of interpreting the Bottom-Left strategy.

We believe that the strategy lies within the recurrent sub-sequences in our abstract representation. As an example, Bottom-Left fills the grid row by row, starting on the same edge as it ends a row. This strategy can be easily described in terms of two simple rules, assuming the game begins at the bottom, most left position: (1) fill in a row, pretty much as a typewriter would write a letter; (2) when (1) is not applicable, go up one row and start all over again (this is in contrast with writing a letter, where we go down, row by row). To recognize these rules, we identify recurrent sub-sequences using Sequitur. This algorithm is capable of getting bigram rules that appear at least twice within a grammar expression.

The results of this abstract representation allow us to decrease the play's space of the videogame. The next step is to apply Sequitur and get recurrent sub-sequences from matches to prove if they can build the strategy of a player; those experiments are presented in the next chapter.

4.1 Building Play's List

The database described in chapter 3 contains all the information to allow us to replay all the matches in order. We represent each play as a list of items with three attributes. The first one is the action; its options are place (P), orient (O), or eliminate(E). The second one is the type of object, either a road (R) or a house (H). The third one is the position; it includes the 2D coordinate in the land and the angle. The player builds his/her match one play at a time; to represent it, we use an ordered list of those plays. Table 4.1 presents the play's list of a partial match with three plays.

Table 4.1: Example of a play's list with three elements.

Order	First	Second	Third
List	P, H, (128, 369, 15^o)	P, R, (182, 297, 90^o)	O, R, (182, 297, 90^o)

Mining a strategy is not straightforward because we need to analyze every play in the match. For each element in the list, the player has three options for the actions, two for the objects, and an infinite amount of possibilities for the position, making the solution space huge. To get the strategy, we need to find relations between all the plays of a match. Since we have an infinite number of possibilities, mining the strategies directly from the list is a difficult task. Furthermore, we have several matches, and we cannot confirm that all the players followed their strategy strictly. Therefore, we need to analyze the matches individually. We propose to abstract the play's list to reduce the play's space.

4.2 Sequence Abstraction

In the first step of abstraction to build a new representation, we aim to section the land. In the research of Cavendeti et al. [12], they divided their map into regions of interest; they are smaller spaces defining regions where most actions happen during a match. We also split our land, although this one is a different case because the videogame does not have specific regions of interest. Instead, we use a 10×10 grid of bricks to divides it. The reason behind this amount is the number of houses that can be inside one brick. Each one has enough space to fit one house and a piece of road that serves as connectivity to the neighborhood, as in figure 4.2.

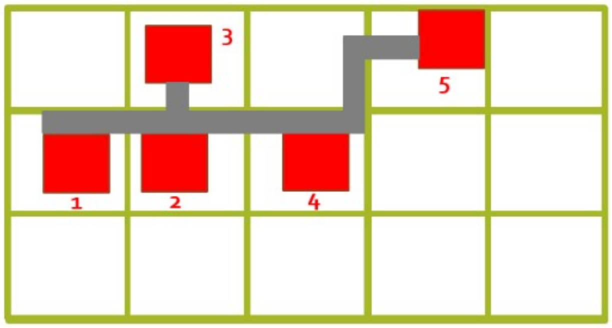


Figure 4.1: Example of a match in 3×4 grid representation. The land is smaller than the one in the videogame.

To transform the play's list using the grid division, first, we will abstract details from the list defined in section 4.1. We think that the strategy of a player is in the order he/she placed the houses. The play's list includes actions for eliminating and orienting objects and connectivity plays. Following our hypothesis, we can remove those plays from the list. The ones remaining are one option for the action, place, and one for the type, house. Furthermore, we believe that the orientation or form of the houses does not determine the strategy, leaving the angle out of the position. An element in the transformed list is $[Place, House, (x, y)]$ where x and y can be any real number; we will represent the list with the position for simplicity.

The transformed list allows us to use the new division of the land to replace the position in the list into grid coordinates. It consists of changing a coordinate (x, y) into (i, j), where x and y are the coordinates in the land, and i and j are integer numbers in discrete space. The grid defines the space for i and j; i is the number of column, and j is the row. We change the position parameter of all elements in the brick's list into the grid coordinates to obtain the abstract sequence. Table 4.2 shows a partial match with five plays in the abstract sequence.

Table 4.2: Abstract sequence of a partial match with five plays.

Plays	First	Second	Third	Fourth	Fifth
Abstract	(1, 6)	(9, 1)	(1, 9)	(9, 9)	(9,4)

4.3 Sequence of Movements

The abstract sequence reduces the details in the play's list. To obtain a strategy, we need to find relations or patterns between a group of plays that describes the behavior of the player during a match. According to our abstracted list, the relations we will find are between positions of house plays. An example in the coordinate system is four consecutive plays, leaving ten units of space from the previous one. With the new representation, we have decreased the infinite amount of possibilities for the position. If a user puts a house in brick, the next play can be 1 of 99 places. To reduce this space even further, we introduce a new relation between two plays called a movement. It consists of two consecutive plays, both in different bricks. The movement describes the displacement of going from the first play to the second one. Figure 4.3 is an abstracted match with five plays, each of them is a circle in the land, and the movements are the green arrows. We defined four different types of movements:

***N* and *S* movements** Represented by the cardinal points, this type of movement is detected when there is a change of row. They are vertical movements.

***E* and *W* movements** Represented by the cardinal points, this type of movement is detected when there is a change of column. They are horizontal movements.

***NW, NE, SW*, and *SE* movements** Represented by the cardinal points, this type of movement is detected when the change in the columns and rows are the same. They are the diagonal movements.

Special movements This type of movement occurs when a player moves from row and column, and he/she reached a border. The name has a $+$ sign depending on the edge reached. For example, a movement $N+W$ happens when it went NW and reached the N border. The change of rows and columns can be different; it can move one row and nine columns, but they must reach a border. There are also movements like $N+W+$, which are when they reached a corner. Figure 4.3 shows an example of a $NW+$.

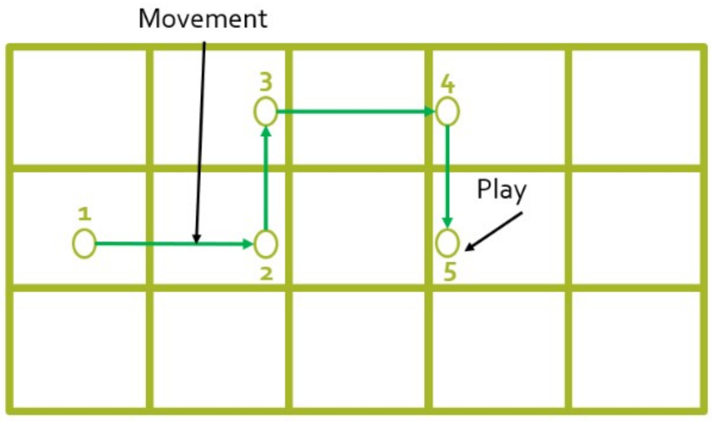


Figure 4.2: Sample abstracted match with five plays. The movements are represented by arrows and the plays by circles.

If a movement does not belong to any of the classes listed above, it will be an unknown one. If a player moves a different number of rows than columns and it does not reach a border, the program does not recognize it. All the strategies that contain an unknown movement cannot be detected using this representation.

To represent a match with movements, we can use the abstract sequence. However, a movement requires two plays; to define a complete match, we need to give a representation to the first and last movements. The first play does not have a preceding one; we use a different attribute to represent it. The videogame stores the first house play as an attribute with three different values: corner, border, or center, depending on the square it lies in. We use this attribute to define the first movement in this representation. The last play does not have a following one; to represent the final movement, we use a null one. However, we believe that the strategy is in the repeated sub-sequences of the match. Since both the first and last movements are unique, they will never be in a repeated sub-sequence. For that reason, we abstracted those two movements. We build the sequence of movements transforming the ordered pairs from the abstract sequence into one of the four types of movements or an unknown one.

The representation of the solutions is oriented to find strategies as Bottom-Left. This strategy is well known for the 2D Bin Packing Problem. In the videogame, a player that uses this strategy begins placing a house on the bottom left corner of the land. He/She continues putting houses left to the previous one. When there is no more space for another one, he/she puts the next one on the bottom most left available position. The player repeats these actions until there is no more space to place another house. Figure 4.3 shows an example of how the Bottom-Left strategy works for this videogame. We can translate the abstracted match of figure 4.3 to the sequence of movements from equation 4.1.

$$E - E - E - E - NW + -E - E - E - E \tag{4.1}$$

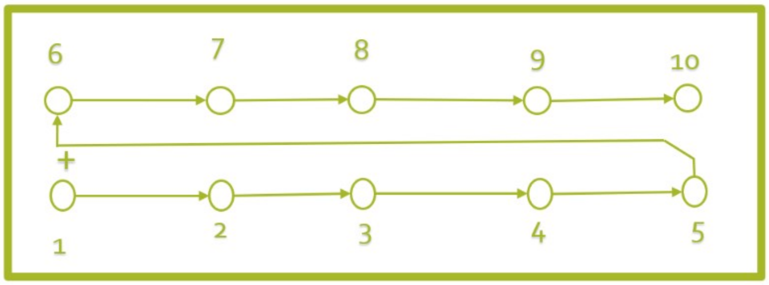


Figure 4.3: Example of a Bottom-Left strategy in an abstracted match. Each circle represents a house; the numbers represent the placement order. The $+$ sign refers to a special movement.

4.4 Sub-Sequences with Sequitur

We can see all the rules that define the Bottom-Left strategy in the movement sequence from equation 4.1. The first one uses E movements until it is not possible. The second one happens only when it cannot apply the first one; it is the $NW+$ movement. With those two rules, we can say that the strategy is Bottom-Left. Our hypothesis suggests that the strategy of a match lies within the recurrent sub-sequences of the movement sequence, like the ones formed with those two rules. Therefore, we need an algorithm capable of extracting such sequences from a movement sequence.

Sequitur is an algorithm that extends the alphabet of a sequence with new grammar rules [52]. The only requirement for the sequence is to be a string. When Sequitur finds a repeated sub-sequence, in a larger one, it forms one grammar rule. The outputs of the algorithm are all the rules and the transformation of the original sequence using those rules. Sequitur follows two principles to generate grammar rules:

Uniqueness: Neither the sequence and the rules can contain repeated digrams.

Utility: A rule must appear at least twice in the sequence or the other rules.

Table 4.3: Example of applying Sequitur to a sequence of characters [34].

Original	bcabcaca
First steps	S → b S → bc S → bca S → bcab
Repeated digram (bc)	S → bcabc
Create rule A	S → AaA rule A → bc
Repeated digram (Aa)	S → AaAa
Create rule B	S → BB rule A → bc rule B → Aa
Utility: rule A	B → bca (A inlined)
Repeated digram (ca)	S → BBca rule B → bca
Create rule C	S → BBca rule B → bC rule C → ca
Final Grammar	S → BBC rule B → bC rule C → ca (deleted: A → bc)

Table 4.3 presents an example of the behavior of Sequitur with a sequence of characters. The algorithm starts with an empty sequence S and adds one character at a time. After adding one, it checks if there is a repeated digram. If that happens, it generates a new rule and substitutes the sub-sequences in S and the other rules with the new one. In some cases, after creating a rule, an existing one can appear just once. It is the case for rule A after adding rule B in table 4.3. By the principle of utility, Sequitur eliminates this type of rule. The algorithm continues until it exhausts the original sequence and returns the rules it found. For example, the rules that remained are B and C, and the sequence changes according to them.

The reason why we chose this algorithm is that all the rules appear at least twice on the original sequence. Each of them has a recurrent sub-sequence. If we use Sequitur for the sequence of equation 4.1, we hope to find sub-sequences that include the rules of Bottom-Left. The experiments that test this hypothesis are in the next chapter.

4.4.1 Sequence Encoding for Sequitur

The sequence of movements that our method provides in section 4.3 fulfills the requirements of the input that Sequitur needs. However, by sending it with movements like SW, Sequitur can mistake them with S or W. Sequitur cannot distinguish the grammar we defined; it understands each character of the sequence as an individual.

Before applying Sequitur to our sequence, we need to encode the movements that are expressed by more than one character. Table 4.4 shows the code for each of them. A sample sequence "$E-E-E-SW+-E$", using the code from table 4.4, transforms into "$EEEGE$". The grammar of this new sequence does not give any hidden meaning to a group of characters as before; the only thing that matters is the order.

Table 4.4: Code for movements with more than one character.

Movement	Code	Movement	Code	Movement	Code	Movement	Code
NW	A	NW+	F	N+W	J	N+W+	O
SW	B	SW+	G	S+W	K	S+W+	P
NE	C	NE+	H	N+E	L	N+E+	Q
SE	D	SE+	I	S+E	M	S+E+	R

4.5 Summary

This chapter showed how to transform a match from the videogame into a sequence. The goal of this is to reduce the search space before performing strategy mining. The first step is to get the information from the database to build the play's list; it contains all the plays in order. Then we make the initial abstraction using a grid division on the land. Such division serves to transform the play space into bricks. To build the abstract sequence, we use the order in which the player placed the houses. To make a second abstraction, we defined four different movements. They consist of the displacement that happens between two consecutive plays. By transforming the abstract sequence into a movement sequence, we can represent the Bottom-Left strategy.

Since we believe that the strategy lies in the recurrent sub-sequences of the match, the chapter ends by describing the algorithm that will allow us to prove this statement. That algorithm is Sequitur; it finds rules composed of sequences from a bigger one. Also, it shows the encoding required to make the algorithm give us the results we need. In the next chapter, we will show how we apply Sequitur to the matches of the players.

Chapter 5

Mining the Strategy of a Player's Match

In this chapter, we used all the matches we collected from the players; we got 113, each of them classified according to their score into high (we call the positive class) or low (we call the negative class). We appealed to Pareto's principle to calculate the threshold that divides both classes. We worked with this dataset in our experiments. Here, we described the results of applying our method for discovering the strategy that a player followed in a match. Our experimentation had two parts: validation and testing.

In the validation phase, we corroborated that our methodology was up to distinguish a particular strategy for dealing with our game, namely: Bottom-Left. To this purpose, we synthesized 13 matches following this strategy, or a simple variant thereof. Note that these synthetic matches are all winning, in that they all yielded a high score, for they were carefully crafted following a good move, as dictated by Bottom-Left. In this part, the synthetic matches substitute the positive class. As shall be seen later on in this Chapter, through Sequitur we successfully identified a collection of rules (patterns) and grammar that reify the Bottom-Left strategy.

In the testing phase, we tested our method using 24 winning matches, gathered out from the database of 113; we divided these matches employing the threshold above mentioned. With the patterns we extracted applying Sequitur, we identified five different strategies, including variants of Bottom-Left. Regarding two of them, we could not find a similar version in the literature for the problems the Housing Development Problem or others alike. The first one starts putting houses in one direction, and when it finds an obstacle, it turns left and continues straight; we called it Snail. The second one focuses on filling the borders first; we named it Rodent.

One of our objectives is to distinguish the strategies that lead to a high score. To meet it, we needed to find the difference between the high and low score matches. It allowed us to know if the worst players used a strategy or if they performed random movements. For this motive, we conducted two tests. The first one evaluated the difference between the patterns Sequitur extracted from the positive and negative classes. In the validating phase, we got that the difference between the synthetic and negative matches was that the second performs movements that do not form repeated sub-sequences; we can see them as random. In the testing phase, we found that players with a high score employed a strategy or a combination. On the other side, for the players with a low score, we could not label their matches with a strategy without entering into specific details due to the excess of random movements.

The second test verifies the patterns we found for the players with a high score. For that, we applied Fisher's exact test and a permutation test. In the validation phase, the tests obtained that the patterns from the strategies we employed more in the synthetic matches are characteristic of the winning players. At the testing phase, we got patterns from three of the strategies we found. Both results suggest that following those strategies might lead to high scores.

The experiments showed that the methodology could extract patterns of the strategies we designed our representation to find, proving that we can describe a strategy using sub-sequences. Using it with matches from players, we rebuilt some strategies, which they employed to get high scores, and the reasons that could lead a player to have a low score. The results encourage us to think that: using patterns, we can create an algorithm that uses strategies from players to give solutions for the Housing Development Problem.

5.1 Matches from the Videogame

The database for the experiment includes all the matches we collected with the videogame. In total, they are 113 with scores from 4 to 24, having 24 as the maximum. For our analysis, we are searching for the strategies of the best players; appealing to Pareto's principle, 20% of our matches have a high score, belonging to the best players, and 80% a low one. With a confidence level of 95% and an error of 8%, we calculate the size of the sample we need as follows:

$$n = \frac{1.95^2 \cdot 0.2 \cdot 0.8}{0.08^2} \approx 97 \tag{5.1}$$

Since our database contains more than 97 users, we can say that the highest score for our videogame is 24 ± 2 with a 95% confidence interval; meaning that it exists the possibility that the strategies we find are not the best.

For a better analysis, we divided the matches into positive and negative classes, depending on their score; doing this, we ensure that the best solutions are the ones with more houses because that is the objective of the videogame. Appealing to Pareto's principle, we defined a threshold to distinguish a high score match. That threshold indicates that a player with a score of at least 19 points has a high score. The amount of positive samples is 24, belonging to the top 20% of the scores.

The worst matches, in the low 20% of the 113, are not considered for the analysis. They performed an average of 6 movements; the best matches did 20. Analyzing the sequences of the lowest score players, we saw a few repeated sub-sequences. Meaning that they did not follow a strategy according to our definition; their movements can be considered random. Those matches can generate noise in the negative class, leading to false conclusions. Therefore, the negative group consists of 60% players above the lowest, and below the best, they are a total of 64. Figure 5.1 shows the division of all the matches we collected.

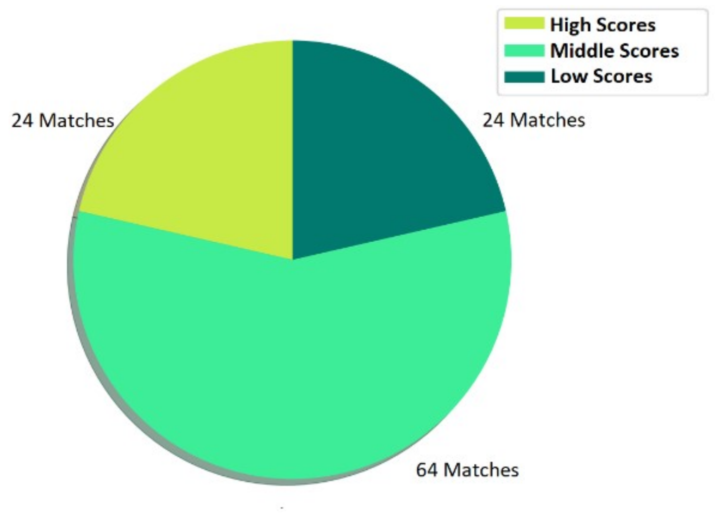


Figure 5.1: Distribution of the matches collected. The high score matches form the positive class, the middle scores the negative class, and the low scores the eliminated class.

Our experimentation is a two-step approach. The first one, called validation, verifies our methodology to identify strategies from repeated sub-sequences using an artificial dataset. The second one, called testing, uses matches in our dataset to define strategies from players with our method. For machine learning, both terms are associated with specific phases of the process. However, the use of them in this study is not related to it; their purpose is to name two different experiments.

5.2 Experiment 1: Validation

This experiment validates the methodology for distinguishing a particular strategy on a match. The objective is to prove that we can describe a strategy using sub-sequences. To confirm the strategy, we need to know how the users played their match; we created synthetic matches for that. They consist of one played following the actions of a known strategy, for example, Bottom-Left. We crafted those matches following the strategy carefully to obtain a high score.

We decided to use strategies similar to Bottom-Left, introduced in chapter 4, for two reasons. First, we designed the representation to be able to detect this strategy; the sequence can describe all the movements in it. Second, we wanted the experiment to simulate players that use different strategies. Bottom-Left has variants by changing the starting position and the directions; figure 5.2 illustrates an example of those variants. Table 5.1 shows the movement sequences of the variants we used of the Bottom-Left of figure 5.2.

The synthetic class consists of 13 matches using a variant of Bottom-Left; table 5.2 shows the distribution by strategy. The reason for this amount is because, to test it with people, we have 24 matches from players. That means we needed 24 synthetic matches or less to prove that, with that amount of positive and negative samples, we can confirm the strategies from the high score players. The times we used a strategy in the synthetic match, as table 5.2 indicates, was to show that we can find strategies that appear few times in the positive class or where is the least amount of matches we need from a strategy to detect it as

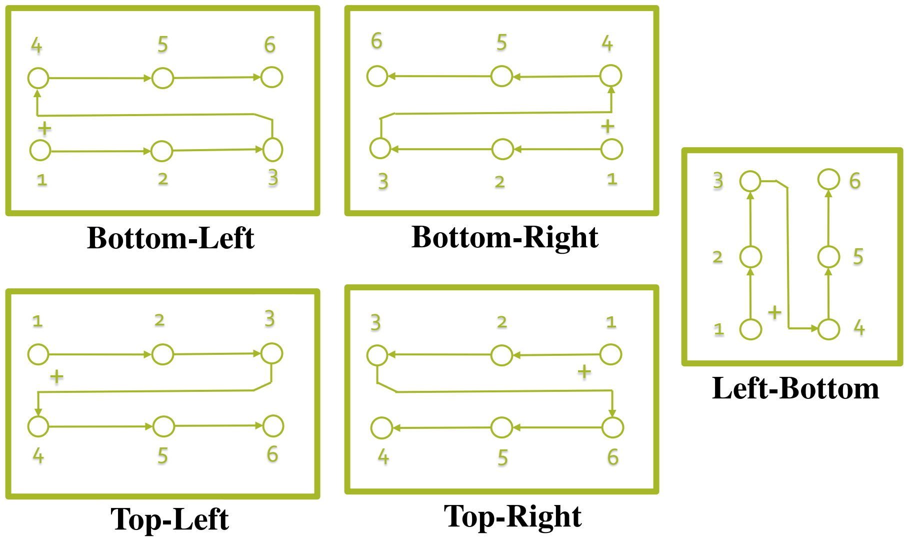


Figure 5.2: Five different variations of the strategy of Bottom-Left.

Table 5.1: Bottom Left style strategies sequences.

Strategy	Sequence
Bottom-Right	W-W-W-W-NE+-W-W-W-W-NE+...
Bottom-Left	E-E-E-E-E-NW+-E-E-E-NW+...
Top-Right	W-W-W-W-SE+-W-W-W-W-SE+...
Top-Left	E-E-E-E-E-SW+-E-E-E-E-E-SW+...
Left-Bottom	N-N-N-N-S+E-N-N-N-N-S+E...

high scoring. Because we were able to find that limit by repeating strategies from 1 to 4 times, and our amount is less than 24, adding more matches does not provide extra information to this experiment.

Since we do not know the strategy of the actual high score matches presented in section 5.1, we needed to substitute the positive class; putting the synthetic matches instead of the positive allowed us to have a control group in the experiment. It will serve us to prove that the strategies we find are the ones with the highest scores. Figure 5.2 shows the substitution in the graph of section 5.1 with the distribution of the matches.

Table 5.2: Amount of matches in the synthetic class by strategy.

Strategy	Matches
Bottom-Right	4
Bottom-Left	3
Top-Right	3
Top-Left	2
Left-Bottom	1

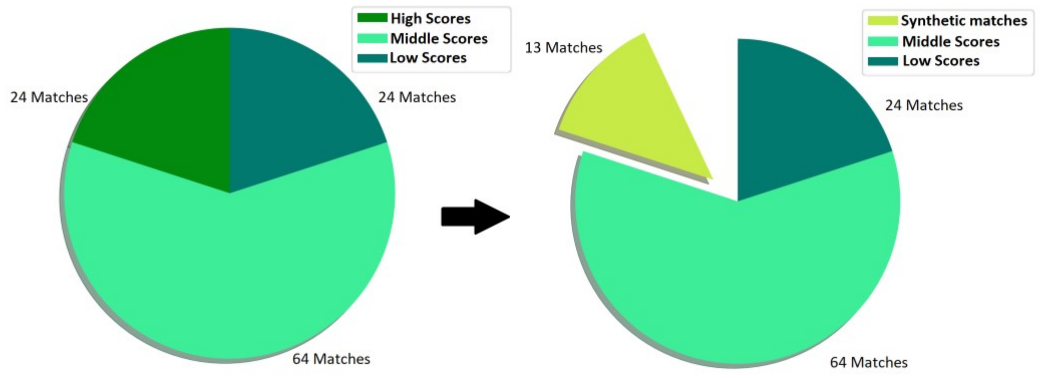


Figure 5.3: Synthetic Matches' Database. The first graph is the same as in figure 5.1. The high score matches belong to the positive class we substitute in the second graph for the synthetic matches. The middle score matches belong to the negative class, and the low score matches are the eliminated class.

In this type of strategy, we can find two main repeated sub-sequences or patterns, presented in table 5.3. The Straight pattern in Bottom-Left are the movements to the left; it can be a single movement or more than one. The Change of Line pattern is the movement when the player cannot place another house to the left. We can also find this pattern with Straight ones surrounding it, for example, $E - NW + -E$.

Table 5.3: Bottom Left strategies patterns.

Pattern	Examples
Straight	E, W, S, N
Change of Line	NE+, NW+, SE+, SW+, S+E

5.2.1 Patterns in Sequitur

A pattern is a brief description of the characteristics of an object [8]. For the sequences we described in section 4.4.1, it is a sub-sequence with at least one movement. The search space, in this case, consists of 113 sequences of at most 24 movements. Our objective is to find repeated sub-sequences. Considering we can have sub-sequences of size from 1 to 24, a match can have at most 300 sub-sequences. Then, the search space is of $24x300x113 = 813600$ sub-sequences in the worst case.

In section 2.3 we described the tests we made to decide to use Sequitur to extract patterns. The version we use of Sequitur is available for python; it receives a string as input and outputs rules for each repeated sub-sequence. Figure 5.2.1 shows all the components of one rule. The first component is the rule; those are the characters Sequitur writes every time it finds that sub-sequence. The second one is the frequency of that rule in the input sequence. The third one is the rule syntaxis; it contains the sub-sequence expressed with all the existing rules. The last one is the pattern; we use them to detect the strategies of the players.

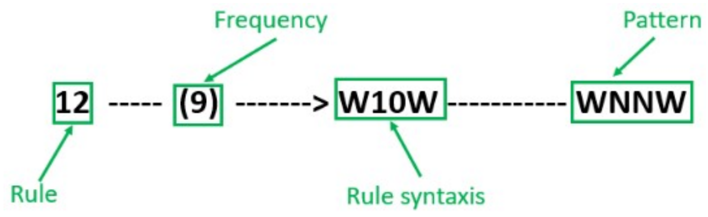


Figure 5.4: Example of a rule from Sequitur.

To use Sequitur, we need sequences of movements from the matches; Sequitur can analyze one at a time. We have to decide on a method to obtain patterns of the synthetic class. There are two options: to apply Sequitur to a sequence composed of all the synthetic matches or to all of them individually. The first method can find patterns that appear once in more than one match, additionally to the ones that happened at least twice in just one. However, in practice, it finds patterns that do not belong to the strategy we called them random; and redundant ones that can be described by other ones. Random patterns probably appear in high score plays, as the following sections mention, to fill empty spaces. Since both methods can define the same amount of strategies, but the second one uses fewer patterns and avoids redundancy and randomness, we applied Sequitur to the individual matches for our experiments.

Individual Matches Method

The method of individual matches applies Sequitur to each sequence in the synthetic group. To obtain the patterns of a match, we use the version of python to apply Sequitur. First, we translate the match using our representation of movements. Then, we give it as input to Sequitur, and it performs the analysis as we described in section 4.4. At last, it outputs all the rules it found, which are the patterns we desire. We repeat this process for all the matches we want to examine.

In total, this method obtained 19 different patterns for the 13 sequences. In table 5.4, we show some of the patterns, avoiding redundant ones; those are the ones that can be explained by others. For example, the pattern $E - E - E$ is redundant because we can explain it with $E - E$. To order them, we use the average frequency in the synthetic class; the average times a pattern appears in all sequences.

With the 8 patterns shown in table 5.4, we can explain all the strategies from the synthetic class; they appear in table 5.1. The first three patterns describe all the Straight patterns from the five strategies. The rest are Change of Line patterns of Top-Right, Bottom-Left, Bottom-Right, Top-Left, and Left-Bottom.

5.2.2 Comparison with Negative class

One of the goals of the study is to find strategies that lead to high scores. The comparison between classes fulfills it, focusing on two objectives. The first one is to detect the difference in the strategies players use in each class. The second is to find the patterns that characterize the best players. It will show us what sub-sequences describe the players in the synthetic group.

Table 5.4: Patterns obtained by applying Sequitur to the synthetic matches individually. They are ordered by the average frequency of appearance in the synthetic class. We show 8 of the 19 we got to avoid redundancy.

Index	Patterns	Avg. Frequency
1	W-W	4.62
2	E-E	3.77
3	N-N	0.85
4	SE+-W-W	0.77
5	E-E-NW+	0.77
6	W-W-NE+	0.46
7	SW+-E-E	0.15
8	N-S+E	0.15

Pattern comparison

In the previous section, we obtained 19 patterns for the synthetic class using the method of individual matches. Now we use the same one to extract patterns from the 64 matches of the negative class; in total, we got 23. If we consider the relation of patterns by matches, the negative class has 0.36 $patterns/match$, while the synthetic has 1.46. It could mean either that the synthetic matches used more strategies or that the negative used more random movements.

We know that the negative matches provide fewer patterns, but to understand the difference between strategies, we analyzed the sequences of the negative class. We can divide them into two groups according to their sequence: random or movements with random changes. The first group performs all of its movements such that there is no repeated sub-sequence. The second group has some repeated sub-sequences surrounded by random movements. Therefore, we cannot explain their match without entering into specific details. Because of those random movements, the negative class has fewer patterns than positive.

We can complement the difference between the two groups for this experiment by calculating the distance among two matches; such distance depends on the distinct movements. To compare two matches, we contrasted the movements in sorted order. Since the negative matches have fewer plays, there are two possibilities, either the movement is different or missing. In the worst case, for the lowest score match, the distance is eight different movements and twelve missings; the best case, which occurs for the highest score match in the negative class, the distance is five different movements and five missings. The movements that differ from the positive group prevent the players in to achieve a high score. According to these tests, a high score depends on following a strategy faithfully, avoiding random plays.

Relevance of Patterns

To find the patterns that characterize the synthetic matches, we perform two tests. The first one is Fisher's exact test, and the second is a permutation test. To build the contingency table for Fisher's exact test, for each pattern, we found how many matches it appears at each class. We present an example of this table for a pattern in table 5.5. Using a significance level of

0.05, 12 out of the 19 patterns approve this test. With them, we can describe three strategies: Bottom-Right, Bottom-Left, and Top-right.

Table 5.5: Contingency table for pattern $W - W - NE+$.

	Positive	Negative	Total
Appeared	3	2	5
Not Appeared	10	62	72
Total	13	64	77

To do the permutation test, we compare the average frequencies of both classes for the 19 section $5.2.1$. First, we calculate the average frequency for each pattern in both classes. Then, with a permutation test, we confirm if there is a significant difference between both groups for the times a player uses a pattern. The permutation test makes a re-sample for both classes; it takes all the frequencies, sorts them, and re-assigns them randomly to the matches. Our test performs $10,000$ permutations; for each one, we compute the difference between the average frequency of both classes. To test if there is a significant difference, we use a 95% two-tail test. We present an example of a permutation test for one pattern in figure 5.2.2.

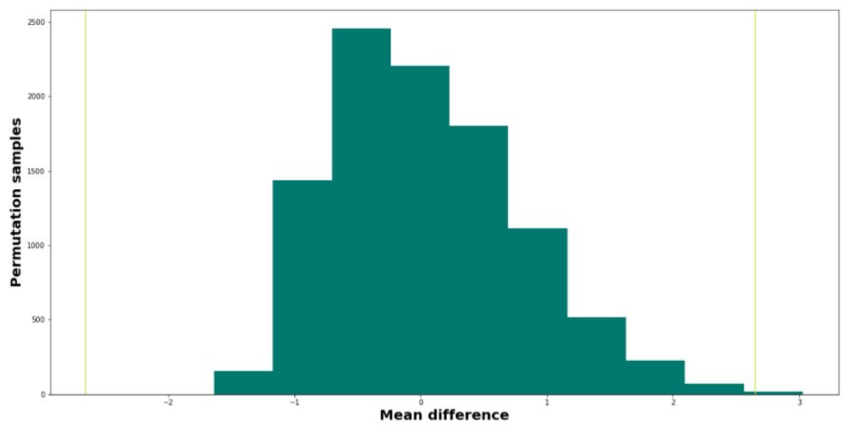


Figure 5.5: Permutation test graph for one pattern. The green lines represent the original difference between the average frequencies.

The test passed 14 patterns of the 19 we had; all of them belong to a strategy we used for the synthetic matches. The first six are the Straight patterns with W or E movements. The other ones are Change of Line patterns that belong to Bottom-Left, Bottom-Right, or Top-Right. The strategies that this test could not detect were used two times or less in the synthetic matches.

Both tests arrive at the same conclusion, even though the first one approves fewer patterns. They suggest that using one of the three strategies more frequently used in the synthetic class will lead to a high score.

5.3 Experiment 2: Testing

After validating our methodology with synthetic matches, we performed the test with real ones. The objective is to detect patterns from the best players to explain their strategy. The patterns we got allowed us to reconstruct several strategies, some of them existing in the literature. The division of the matches is the one we explained in section 5.1. The experiment concludes by comparing the negative class with the positive as in the validation experiment.

5.3.1 Patterns Applying Sequitur

To obtain the patterns, we applied Sequitur to all the 24 matches from the positive class, as we did in the validation phase; we got a total of 24 patterns. To order them, we calculated the average frequency in the positive class. We present them in tables 5.6 and 5.7

Table 5.6: Most frequent patterns obtained applying Sequitur to the positive class. Their order is according to the average frequency in the positive matches.

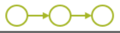
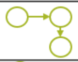
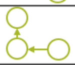
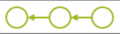
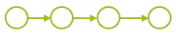
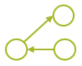
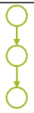
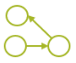
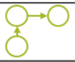
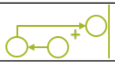
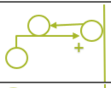
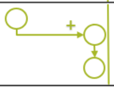
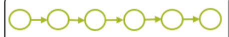
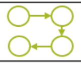
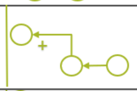
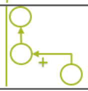
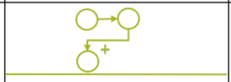
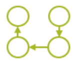
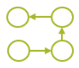
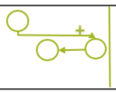
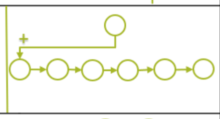
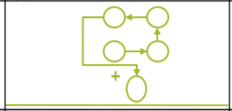
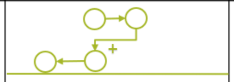
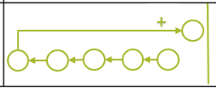
Index	Patterns	Graph representation	Avg. Frequency
1	E-E		2.68
2	E-S		1.64
3	W-N		1.60
4	W-W		1.52
5	E-E-E		1.36
6	W-NE		0.72
7	S-S		0.68
8	E-NW		0.52
9	N-W		0.48
10	W-NE+		0.40
11	NE+-W		0.40
12	SE+-S		0.40

Table 5.7: Less frequent patterns obtained applying Sequitur to the positive class. Their order is according to the average frequency in the positive matches.

Index	Patterns	Graph representation	Avg. Frequency
13	E-E-E-E-E		0.36
14	E-S-W		0.36
15	W-NW+		0.36
16	NW+-N		0.36
17	E-S+W		0.32
18	S-W-N		0.28
19	E-N-W		0.20
20	SE+-W		0.20
21	SW+-E-E-E-E-E		0.12
22	E-N-W-S+E		0.12
23	E-S+W-W		0.08
24	W-W-W-W-NE+		0.08

Using patterns from tables 5.6 and 5.7 we identified five strategies, presented in figure 5.3.1. For the first two, we could not find a similar version in the literature; we named them Snail and Rodent. To illustrate, in the Snail strategy, some matches used patterns 1, 14, 4, and 3. It starts placing houses in one direction until it reaches an obstacle, then it makes a turn to continue putting houses. It stops when there is no space where to turn. In the second strategy, Rodent, players performed consecutively patterns 12 and 7, or 23, and 4. It starts filling the space in the border with houses; then, it fills the center. The other three strategies are similar to Bottom-Left. To make an example, some players used patterns 13 and 21.

Analyzing the sequences of the 24 matches, we discovered that they use the five strategies to build their solution. There are cases in which they followed a strategy strictly; in other ones, they combined strategies or complemented them with other movements. Nevertheless, we can classify them manually with one or two labels of the five we defined.

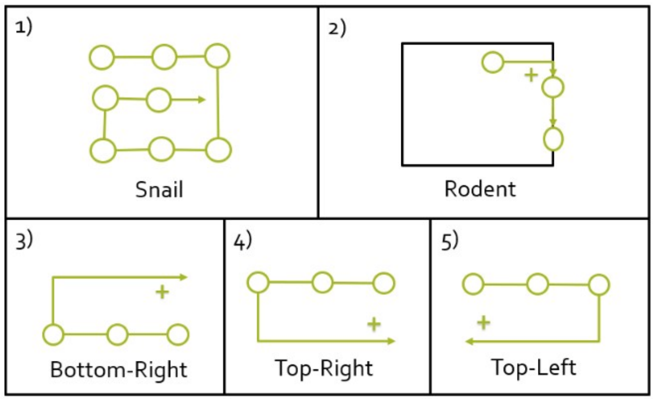


Figure 5.6: Strategies reconstructed with patterns from players.

5.3.2 Comparison with Negative class

The validation phase allowed us to test the methodology for comparing two classes. To complete our goal of finding strategies that lead to high scores, we need to perform the tests with real players. This section follows the same structure as the one in the validation phase.

Pattern comparison

To perform the pattern comparison, we need to apply Sequitur to both classes. The positive patterns come from section 5.3.1, they are 24. The negative ones are the same as the validation phase because we are using the same matches; they are 23. The first thing we noticed is that the average amount of patterns per match from the positive group is greater than the negative; they have values of 1.0 and 0.36.

Regarding the distance between two matches from the two classes, we have two cases. The first one is when we compare a match from the negative group with one that uses one of the variants of Bottom-Left in the positive group. In this case, the distance is the same as in the validation experiment. In the second case, other matches in the positive group have all the movements different than the ones in the negative group.

Analyzing the sequences of the positive group, we saw that some people were faithful to one strategy; others used a combination of two or one that we could not identify with this methodology. Additionally, some of them performed random movements, probably to fill empty spaces. According to the inspection of the negative sequences from section 5.2.1, the negative class has a group of matches that used a combination of patterns with random movements. What makes a player get a high score, according to this test, lies in the use of one or two strategies of the five we found. Additionally, random movements can complement the strategy, but its use must be restricted to one or two consecutive movements.

Relevance of Patterns

In this part of the experiment, we want to find the strategies that characterize the best players. The process is equal to the one of experiment 1; we perform Fisher's exact test and a permutation test with the 24 patterns from section 5.3.1.

Fisher's exact test approved 6 patterns; with them, we can describe completely one strategy, Snail. Besides, some of the patterns belong to the strategies of Rodent and Bottom-Right. This test suggests that these two strategies can give good results when used to complement other strategies. For example, the Rodent strategy needs to do something different to fill the center of the land. With the permutation test, 12 patterns were approved; they belong to three of the five strategies: Snail, Bottom-Right, and Rodent.

The remaining two strategies that neither of the tests found might be present in both classes. However, players in the positive one could add different movements to complement the strategy, causing them to obtain a higher result than the players in the negative class. Another possibility is that the number of players in the negative class that use those strategies is greater than the ones in the positive, obtaining a frequency similar to the few we have in a positive group. To prove the correct case, we need more matches with a high score. However, we can confirm that using any of these three strategies will lead to a high score.

5.4 Summary

Using our videogame, we obtained information about matches from players, and we divided them into two classes for the study, positive and negative. This chapter presented our methodology to build strategies using patterns; it consisted of three steps. First, we obtained patterns from the positive matches using Sequitur; second, we got patterns from the negative matches and compared both classes; third, we applied a Fisher's exact test and permutation test to each of the patterns from the positive group.

The study has two experiments, validation, and testing. The validation phase used synthetic matches, instead of the positive class, to confirm that the results we got are related to the strategies they used; proving that we can detect a strategy using repeated sub-sequences. The testing phase applied the methodology to extract patterns from players and analyzed the difference between people that got a high score with the ones that got a low one. The results encourage us to think that we can build algorithms that use patterns to solve the problem using a player's strategy.

Chapter 6

Conclusions and Future Work

This chapter's objective is to generalize the work made in the study, mentioning the strengths and weaknesses of the model for the Housing Development Problem and the methodology for pattern extraction. It emphasizes the results obtained from chapter 5, recognizing strategies used by players. Additionally, it answers the research questions from chapter 1.4.

6.1 Videogame Crowdsourcing Model

One of the most important decisions for this study is the selection of the problem for the Videogame Crowdsourcing. That defines the model to follow in the videogame and the solutions you will find. It is essential to understand the problem to make the solutions of players useful. In our case, we can only work to optimize the number of houses due to our initial definition.

Architecture problems involve many more features and objects different than houses, as we said in chapter 3. The Videogame Crowdsourcing model from the study does not evaluate aspects like aesthetics, cost, or environmental impact. The reason is that players need to understand the videogame, and adding more features makes it more complicated. To evaluate the comprehension of a videogame, we would need to convey a survey with a group of people. Since the videogame was not the main goal of this study, and due to time restrictions, we could not make an exhaustive evaluation of this aspect. For this purpose, we were able to collect several matches that serve us as feedback. Analyzing them, we saw that some players understood the rules while others could not get a score; the ones that understood belong to the 113 we mentioned in chapter 4; the others were not reported. It can mean either that the videogame is hard to understand, or it is directed to a group of people.

As seen in chapter 3, the proposed model for the Housing Development Problem fits in a videogame and evaluates the aspect we desired. The objective is to optimize the space occupied by houses while maintaining the connections between them. We show this in the definition of a valid answer for the videogame. When solving a videogame, the players get feedback with the score, and it increases as the solution has more houses. We selected this function to work with a single objective model. For projects where the number of houses of each available design and remaining free space do not matter, this model fits perfectly. However, if we would like to minimize the remaining space, the number of houses of distinct models, the area occupied by road, or another objective, this model will not work. The reason is that it does not evaluate the player's performance on those aspects; the score depends on the total amount of houses. The evaluation function has to be adjusted to make it multi-objective.

For the data storage in the Videogame Crowdsourcing, we selected a service from Firebase. We can conclude that it is reliable for storing all the information from a web page in real-time. It collected information since a player registered to the time it left. With the database, we knew what happened in a match or if the player could not finish the videogame. That allowed us to reconstruct the 113 matches from players and the synthetic we made. For our Videogame Crowdsourcing, it fulfilled the objective of storing all the information about the people that played our videogame.

6.2 Task of Pattern Recognition to Define Strategies

In chapter 4, we explained the difficulties of mining patterns from the information extracted directly from every match. Instead, we proposed a sequence representation, reducing the play's search space; it assumes that all the house plays have the same characteristics, except for the location, and abstracts correction and connection plays. Such sequence representation succeeded in maintaining the details of the order of the plays, allowing us to search for the strategy of the players. Nevertheless, like any abstraction, we might lose information that causes our solution to differ from reality. There are many ways to interpret a match of this videogame; we focused our representation on finding strategies similar to Bottom-Left. This decision relied on the similarities that our videogame shares with the 2D Bin Packing Problem. In our validation experiment of chapter 5, we included 13 synthetic matches of the videogame using this type of strategy. From all of them, we were able to extract the sequence of movements in order and explain the strategy. Then this representation is capable of describing all the plays of these strategies or other similar.

Depending on the representation, the computational tool that obtains better descriptors of the strategies can variate. In our case, it was essential to keep the order of the sequences. As we mentioned in section 2.3, from the patter miners we tried, Sequitur gave better results. The reason is that it is capable of extracting all the repeated sub-sequences from a match using our representation. We can prove this with the validation experiment. Sequitur used the 13 synthetic matches and got all the Straight and Change of Line patterns that describe the five strategies we used. Nevertheless, a change in the representation can allow the exploration of the idea of using different algorithms.

In chapter 5, we defined the methodology followed in the study. First, we did the validation experiment to prove that we can detect strategies finding repeated sub-sequences; here, we got the five strategies of the synthetic matches. Then, in the comparison experiment in the validation phase, we saw that people with low scores do not follow a strategy strictly as we made it in the synthetic matches. The results showed that the negative class presented fewer patterns than the synthetic group. When we analyzed the sequences individually, we found that players from the negative group made more random movements, resulting in fewer patterns. More importantly, when searching for the patterns that describe the synthetic class, the results confirmed that strategies with less amount of matches are not detected. With Fisher's exact test and the permutation test, we found patterns that characterize the synthetic class;

those describe three of the five strategies. We think that the dataset needs to be balanced to detect all possible strategies that lead to a high score.

Working with matches from real players, we got 24 patterns, applying Sequitur, which described five strategies they use. Among them, we defined Snail and Rodent, which we could not find in the literature. All of the patterns we found depend on our representation, which divides the original land into sections, restricting the use of those strategies for instances that have a polygon of four vertices as the available land for construction. The reason is that the spaces in that land that define the movements are blocks of the same size. If the land was different, we could divide it into squares and apply the strategy to each of them, or define another division to get new strategies.

As we mentioned before, we oriented our representation to detect all the plays of a Bottom-Left strategy or another with similar movements. The assumption behind this decision is that the best strategies have these movements. However, we might encounter an unknown movement or patterns that seem to be random. In this case, a different representation can help us to describe the plays that ours cannot and complement the information about the strategies we found in this study.

Comparing the positive and negative classes, we found that players in the positive group performed more repeated sub-sequences, resulting in more patterns. Players in the negative class made more random patterns. Regarding the sequences, we saw that positive players do not follow a strategy strictly, as the synthetic matches. Some people focused on one of the five strategies we found, while others combined some of them. Additionally, some matches performed few random movements, probably to fill empty spaces left by their strategy. An interesting finding was that users executed strategies similar to the ones in the literature, answering one of our research questions. From the 24 patterns Sequitur found, some of them belong to strategies like Bottom-Left, and the sequences we got confirmed it. Players may or not be aware that they are applying this strategy; if they do, we think it is because they have used it in another similar problem and got good results.

Another of our research questions asks about the difference between the strategy of players with high and low scores. In the testing experiment, we showed that people in the positive group use one or two strategies of the five we found, some of them complementing the strategy with random movements. Fisher's exact test and the permutation test approved patterns that described the strategies of Snail, Bottom-Right, and Rodent, suggesting that they lead to a high score. However, our dataset consisted of just 113 matches; without a big dataset, we cannot discard the possibility of other strategies leading to a high score, but we can assure that these three strategies will result in one. The main difference in the matches that defines the score is the path people follow to the solution. A good one will select one or two strategies and fill the empty spaces the strategies leave with random movements or another strategy.

It is important to note that the objective of this study was to find the strategies that lead to the best solutions. For a different application, it can be relevant to know the path that people took to get a low score. The methodology to achieve it is the same, but instead of finding patterns for high score players, we need to apply Sequitur to the low score players and make the analysis. Nevertheless, in this study, negative cases served us to contrast the matches with the positive ones.

6.3 Future Work

The task of finding strategies from a videogame has a lot of opportunities. From modeling the problem to analyzing the information, there are many options that this study did not explore. First, the videogame can add features to make a more exact simulation of the Housing Development Problem. Without producing a more difficult videogame, we can include additional objects that residential areas may have, like pools, parks, among others; our model can variate the figure of the available land for that. For example, if an architect wishes to add a park in the residential area, this model can put it as a fixed object in the initial state and remove that piece of land for the available space for the construction. This feature can present different levels of this videogame to the players. Although, research that focuses on this topic should consider that players must understand the videogame to enjoy it and participate. A survey that evaluates the aspect of comprehension could guide researchers to design a videogame that contains as most features as possible while people enjoy playing the videogame. Also, the model can change to adapt to other problems similar to the 2D Bin Packing problem. It could result in a fascinating comparison to see if players use the same strategy for different problems with the same model.

An interesting challenge in this study was to design the representation and find an algorithm that could extract useful information. Distinct methodologies could focus on representations for finding strategies different from Bottom-Left. That might explain other movements that we could not define as patterns, even describe the ones that include an unknown movement. Using the same representation but adding the plays we abstracted or changing it into a different one could lead to different results. Comparing distinct representations and algorithms to the one we proposed could complement the information of the strategies we found of the players and get new results.

One thing this study missed was a big database of matches to ensure that the strategies are the best ones. For that purpose, research can design a publicity campaign for the videogame using social media. Nevertheless, the results encourage us to think that an algorithm that uses patterns from players could get a solution for the Housing Development Problem. The next step of this research is to create an algorithm based on the strategies we found and compare the algorithm with existing solutions for the Housing Development Problem.

Bibliography

[1] Abeysooriya, R. P., Bennell, J. A., and Martinez-Sykora, A. Jostle heuristics for the 2D-irregular shapes bin packing problems with free rotation. *International Journal of Production Economics 195* (jan 2018), 12–26.

[2] Agis, R. A., Gottifredi, S., and García, A. J. An event-driven behavior trees extension to facilitate non-player multi-agent coordination in video games. *Expert Systems with Applications 155* (oct 2020).

[3] Bahrehmand, A., Batard, T., Marques, R., Evans, A., and Blat, J. Optimizing layout using spatial quality metrics and user preferences. *Graphical Models 93* (sep 2017), 25–38.

[4] Bakkes, S. C., Spronck, P. H., and van Lankveld, G. Player behavioural modelling for video games. *Entertainment Computing 3*, 3 (aug 2012), 71–79.

[5] Belhadi, A., Djenouri, Y., Lin, J. C. W., and Cano, A. A general-purpose distributed pattern mining system. *Applied Intelligence 50*, 9 (sep 2020), 2647–2662.

[6] Beyaz, M., Dokeroglu, T., and Cosar, A. Robust hyper-heuristic algorithms for the offline oriented/non-oriented 2D bin packing problems. *Applied Soft Computing 36* (nov 2015), 236–245.

[7] Brabham, D. C. *Crowdsourcing*. The MIT Press Essential Knowledge Series, 2013.

[8] Cañete-Sifuentes, L., Monroy, R., Medina-Pérez, M. A., Loyola-González, O., and Vera Voronisky, F. Classification Based on Multivariate Contrast Patterns. *IEEE Access 7* (2019), 55744–55762.

[9] Cao, D., and Kotov, V. M. A best-fit heuristic algorithm for two-dimensional bin packing problem. In *Proceedings of 2011 International Conference on Electronic and Mechanical Engineering and Information Technology, EMEIT 2011* (2011), vol. 7, pp. 3789–3791.

[10] Carlini, E., and Lulli, A. Analysis of Movement Features in Multiplayer Online Battle Arenas. *Journal of Grid Computing 17*, 1 (mar 2019), 45–57.

[11] Carlini, E., Lulli, A., and Ricci, L. TRACE: Generating traces from mobility models for distributed virtual environments. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2017), vol. 10104 LNCS, Springer Verlag, pp. 272–283.

[12] CAVADENTI, O., CODOCEDO, V., BOULICAUT, J.-F., AND KAYTOUE, M. What did i do wrong in my moba game?:mining patterns discriminating deviant behaviours. In *Proceedings of the 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)* (Montreal, QC, Canada, 2016), IEEE Computer Society, pp. 662–671.

[13] CHAN, C. S. Cognitive processes in architectural design problem solving. *Design Studies 11*, 2 (1990), 60–80.

[14] CHITTARO, L., RANON, R., AND LERONUTTI, L. VU-Flow: A visualization tool for analyzing navigation in virtual environments. *IEEE Transactions on Visualization and Computer Graphics 12*, 6 (nov 2006), 1475–1485.

[15] DÓSA, G. The tight bound of first fit decreasing bin-packing algorithm Is FFD(I) 11/9OPT(I) + 6/9. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2007), vol. 4614 LNCS, Springer Verlag, pp. 1–11.

[16] DRACHEN, A., AND CANOSSA, A. Analyzing spatial user behavior in computer games using geographic information systems. In *MindTrek 2009 - 13th International Academic MindTrek Conference: Everyday Life in the Ubiquitous Era* (New York, New York, USA, 2009), ACM Press, pp. 182–189.

[17] ESTRADA, L. E. P., GROEN, D., AND RAMIREZ-MARQUEZ, J. E. A serious video game to support decision making on refugee aid deployment policy. *Procedia Computer Science 108* (2017), 205–214.

[18] FAKERE, A. A., ARAYELA, O., AND FOLORUNSO, C. O. Nexus between the participation of residents in house design and residential satisfaction in Akure, Nigeria. *Frontiers of Architectural Research 6*, 2 (jun 2017), 137–148.

[19] FERNANDEZ-ARES, A., MORA, A. M., MERELO, J. J., GARCIA-SANCHEZ, P., AND FERNANDES, C. Optimizing player behavior in a real-time strategy game using evolutionary algorithms. In *2011 IEEE Congress of Evolutionary Computation, CEC 2011* (2011), pp. 2017–2024.

[20] FERNÁNDEZ-ARES, A. J., GARCÍA-SÁNCHEZ, P., MORA, A. M., AND MERELO, J. J. Adaptive bots for real-time strategy games via map characterization. In *2012 IEEE Conference on Computational Intelligence and Games, CIG 2012* (2012), pp. 417–423.

[21] GABBE, C. Local regulatory responses during a regional housing shortage: An analysis of rezonings in Silicon Valley. *Land Use Policy 80* (jan 2019), 79–87.

[22] HANSSON, A. G. Promoting planning for housing development: What can Sweden learn from Germany? *Land Use Policy 64* (may 2017), 470–478.

[23] HOMAYOUNI, H. A survey of computational approaches to space layout planning (1965-2000).

[24] HOWE, J. The rise of crowdsourcing. *Wired Magazine 14.06* (2006), 1–5.

[25] HUANG, J., SHEN, G. Q., AND ZHENG, H. W. Is insufficient land supply the root cause of housing shortage? Empirical evidence from Hong Kong. *Habitat International 49* (oct 2015), 538–546.

[26] IVANJKO, T. Crowdsourcing image descriptions using gamification: a comparison between game-generated labels and professional descriptors. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (may 2019), IEEE, pp. 537–541.

[27] JAKOBS, S. On genetic algorithms for the packing of polygons. *European Journal of Operational Research 88*, 1 (jan 1996), 165–181.

[28] JOHNS, M. B., ROSS, N. D., MAHMOUD, H. A., KEEDWELL, E. C., WALKER, D. J., AND SAVIC, D. A. Augmented evolutionary intelligence: Combining human and evolutionary design for water distribution network optimisation. In *GECCO 2019 - Proceedings of the 2019 Genetic and Evolutionary Computation Conference* (New York, NY, USA, jul 2019), Association for Computing Machinery, Inc, pp. 1214–1222.

[29] KARPOV, I. V., SCHRUM, J., AND MIIKKULAINEN, R. Believable bot navigation via playback of human traces. In *Believable Bots: Can Computers Play Like People?*, vol. 9783642323232. Springer-Verlag Berlin Heidelberg, oct 2012, pp. 151–170.

[30] KARYKOW, A., ROUMANIS, G., KAM, A., KWAK, D., LEUNG, C., WU, C., ZAROUR, E., SARMENTA, L., BLANCHETTE, M., AND WALDISPÃHL, J. Phylo: A citizen science approach for improving multiple sequence alignment. *PLOS one 7*, e31362 (2012).

[31] KHAMENEH, N. Y., AND GUZDIAL, M. Entity embedding as game representation. In *2020 Experimental AI in Games Workshop* (October 2020).

[32] KIM, J. S., GREENE, M. J., ZLATESKI, A., LEE, K., RICHARDSON, M., TURAGA, S. C., PURCARO, M., BALKAM, M., ROBINSON, A., BEHABADI, B. F., CAMPOS, M., DENK, W., AND SEUNG, H. S. Spaceâtime wiring specificity supports direction selectivity in the retina. *Nature 509* (2014), 331–336.

[33] KRACHT, C. L., JOSEPH, E. D., AND STAIANO, A. E. Video Games, Obesity, and Children. *Current Obesity Reports 9*, 1 (mar 2020), 1–14.

[34] LATENDRESSE, M. Masquerade detection via customized grammars. In *Lecture Notes in Computer Science* (2005), vol. 3548, Springer Verlag, pp. 141–159.

[35] LIGGETT, R. S. The quadratic assignment problem: an analysis of applications and solution strategies. *Environment and Planning B: Planning and Design 7*, 2 (1980), 141–162.

[36] LITMAN, T., ALEX, T., AND LITMAN, E. Affordable-accessible housing in a dynamic city: Why and how to increase affordable housing development in accessible locations, victoria transport policy institute, 2010.

[37] LIU, D., AND TENG, H. An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles. *European Journal of Operational Research 112*, 2 (jan 1999), 413–420.

[38] LIU, Q., ZENG, J., ZHANG, H., AND WEI, L. A heuristic for the two-dimensional irregular bin packing problem with limited rotations. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (sep 2020), vol. 12144 LNAI, Springer Science and Business Media Deutschland GmbH, pp. 268–279.

[39] LIU, S., LOUIS, S. J., AND NICOLESCU, M. Using CIGAR for finding effective group behaviors in RTS game. In *IEEE Conference on Computatonal Intelligence and Games, CIG* (2013).

[40] LÓPEZ-CAMACHO, E., TERASHIMA-MARÍN, H., AND ROSS, P. A hyper-heuristic for solving one and two-dimensional bin packing problems. In *Genetic and Evolutionary Computation Conference, GECCO'11 - Companion Publication* (New York, New York, USA, 2011), ACM Press, pp. 257–258.

[41] LOYOLA-GONZÁLEZ, O., MEDINA-PÉREZ, M. A., MARTÍNEZ-TRINIDAD, J. F., CARRASCO-OCHOA, J. A., MONROY, R., AND GARCÍA-BORROTO, M. PBC4cip: A new contrast pattern-based classifier for class imbalance problems. *Knowledge-Based Systems 115* (jan 2017), 100–109.

[42] MACHCHHAR, J., AND ELBER, G. Dense packing of congruent circles in free-form non-convex containers. *Computer Aided Geometric Design 52-53* (mar 2017), 13–27.

[43] MARKATOU, M. Urban Planning and Greening Practices: A Case For Neighborhood Development in a Typical Urban Area. *Journal of Environmental Science and Engineering 9* (2020), 189–199.

[44] MAVANDADI, S., FENG, S., YU, F., DIMITROV, S., YU, R., AND OZCAN, A. Biogames: A platform for crowd-sourced biomedical image analysis and telediagnosis. *Games for Health Journal 1* (2012), 373–376.

[45] MEDJDOUB, B., AND YANNOU, B. Dynamic space ordering at a topological level in space planning. *Artificial Intelligence in Engineering 15*, 1 (jan 2001), 47–60.

[46] MEXICANO SANTOYO, A., PÉREZ ORTEGA, J., REYES SALGADO, G., ALMANZA ORTEGA, N. N., AND ORTEGA, N. N. A. Characterization of Difficult Bin Packing Problem Instances oriented to Improve Metaheuristic Algorithms. *Computación y Sistemas 19*, 2 (jun 2015), 295–308.

[47] MICHALEK, J., CHOUDHARY, R., AND PAPALAMBROS, P. Architectural layout design optimization. *Engineering Optimization 34*, 5 (jan 2002), 461–484.

[48] MOURA, D., EL-NASR, M. S., AND SHAW, C. D. Visualizing and understanding players' behavior in video games: Discovering patterns and supporting aggregation and

comparison. In *ACM SIGGRAPH 2011 Game Papers, SIGGRAPH'11* (New York, New York, USA, 2011), ACM Press, p. 1.

[49] MOURATIDIS, K. Neighborhood characteristics, neighborhood satisfaction, and well-being: The links with neighborhood deprivation. *Land Use Policy 99* (dec 2020), 104886.

[50] MUÑOZ, J., GUTIERREZ, G., AND SANCHIS, A. Towards imitation of human driving style in car racing games. In *Believable Bots: Can Computers Play Like People?*, vol. 9783642323232. Springer-Verlag Berlin Heidelberg, oct 2012, pp. 289–313.

[51] NEGREVERGNE, B., TERMIER, A., ROUSSET, M. C., AND MÉHAUT, J. F. Para Miner: A generic pattern mining algorithm for multi-core architectures. *Data Mining and Knowledge Discovery 28*, 3 (may 2014), 593–633.

[52] NEVILL-MANNING, C. G., AND WITTEN, I. H. Identifying Hierarchical Structure in Sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research 7* (sep 1997), 67–82.

[53] ONTAÑÓN, S., MISHRA, K., SUGANDH, N., AND RAM, A. Case-based planning and execution for real-time strategy games. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2007), vol. 4626 LNAI, Springer Verlag, pp. 164–178.

[54] OPOKU, R. A., AND ABDUL-MUHMIN, A. G. Housing preferences and attribute importance among low-income consumers in Saudi Arabia. *Habitat International 34*, 2 (apr 2010), 219–227.

[55] ORTEGA, J., SHAKER, N., TOGELIUS, J., AND YANNAKAKIS, G. N. Imitating human playing styles in Super Mario Bros. *Entertainment Computing 4*, 2 (apr 2013), 93–104.

[56] PFAU, J., LIAPIS, A., VOLKMAR, G., YANNAKAKIS, G. N., AND MALAKA, R. Dungeons & Replicants: Automated Game Balancing via Deep Player Behavior Modeling. In *2020 IEEE Conference on Games (CoG)* (aug 2020), IEEE, pp. 431–438.

[57] PFAU, J., SMEDDINCK, J. D., AND MALAKA, R. Towards deep player behavior models in MMORPGs. In *CHI PLAY 2018 - Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play* (oct 2018), Association for Computing Machinery, Inc, pp. 341–351.

[58] ROSS, N., KEEDWELL, E., AND SAVIC, D. Human-derived heuristic enhancement of an evolutionary algorithm for the 2d bin-packing problem. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (sep 2020), vol. 12270 LNCS, Springer Science and Business Media Deutschland GmbH, pp. 413–427.

[59] SVATOŇ, V., MARTINOVIČ, J., SLANINOVÁ, K., AND BUREŠ, T. Improving strategy in robot soccer game by sequence extraction. In *Procedia Computer Science* (jan 2014), vol. 35, Elsevier B.V., pp. 1445–1454.

[60] SYNNAEVE, G., AND BESSIÈRE, P. A Bayesian model for opening prediction in RTS games with application to StarCraft. In *2011 IEEE Conference on Computational Intelligence and Games, CIG 2011* (2011), pp. 281–288.

[61] TANNA, P., AND GHODASARA, D. Y. Using Apriori with WEKA for Frequent Pattern Mining. *International Journal of Engineering Trends and Technology 12*, 3 (jun 2014), 127–131.

[62] TERASHIMA-MARÍN, H., ROSS, P., FARÍAS-ZÁRATE, C. J., LÓPEZ-CAMACHO, E., AND VALENZUELA-RENDÓN, M. Generalized hyper-heuristics for solving 2D Regular and Irregular Packing Problems. *Annals of Operations Research 179*, 1 (sep 2010), 369–392.

[63] TESFAYE, A. Problems and prospects of housing development in Ethiopia, 2007.

[64] UNO, T., KIYOMI, M., AND ARIMURA, H. LCM ver. 2: Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets. In *Proceedings of IEEE ICDM'04 Workshop* (2004).

[65] VAN HOORN, N., TOGELIUS, J., WIERSTRA, D., AND SCHMIDHUBER, J. Robust player imitation using multiobjective evolution. In *2009 IEEE Congress on Evolutionary Computation, CEC 2009* (2009), pp. 652–659.

[66] VON AHN, L. Games with a purpose. *IEEE Computer Magazine 39* (2006), 92–94.

[67] VON AHN, L., LIU, R., AND BLUM, M. Peekaboom: a game for locating objects in images. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Montreal, Canada, 2006), CHI, pp. 1767–1774.

[68] WANG, L., AND JIANG, T. On the complexity of multiple sequence alignment. *Journal of computational biology 1* (1994), 337–348.

[69] WAZNY, K. "Crowdsourcing" ten years in: A review. *Journal of Global Health 7*, 2 (dec 2017), 20601.

[70] WEBER, B. G., AND MATEAS, M. A data mining approach to strategy prediction. In *CIG2009 - 2009 IEEE Symposium on Computational Intelligence and Games* (2009), pp. 140–147.

[71] WONG, S. S., AND CHAN, K. C. EvoArch: An evolutionary algorithm for architectural layout design. *CAD Computer Aided Design 41*, 9 (sep 2009), 649–667.

[72] XIN, K., ZHANG, S., WU, X., AND CAI, W. Reciprocal crowdsourcing: Building cooperative game worlds on blockchain. In *2020 IEEE International Conference on Consumer Electronics (ICCE)* (2020), pp. 1–6.

[73] YANNAKAKIS, G. N., AND TOGELIUS, J. A Panorama of Artificial and Computational Intelligence in Games. *IEEE Transactions on Computational Intelligence and AI in Games 7*, 4 (dec 2015), 317–335.

[74] YE, Y., GUANGRUI, F., AND SHIQI, Q. An algorithm for judging points inside or outside a polygon. In *Proceedings - 2013 7th International Conference on Image and Graphics, ICIG 2013* (2013), pp. 690–693.

[75] ZAWIDZKI, M., AND SZKLARSKI, J. Multi-objective optimization of the floor plan of a single story family house considering position and orientation. *Advances in Engineering Software 141* (mar 2020), 102766.

[76] ZHENG, H., AND REN, Y. Architectural layout design through simulated annealing algorithm. In *Proceedings of the 25th International Conference on Computer-Aided Architectural Design Research in Asia* (Bangkok, Thailand, 2020), Association for Computer-Aided Architectural Design Research in Asia, pp. 275–284.

Curriculum Vitae

Master student was born in Mexico City, México, on September 13, 1994. He earned the Biomedical Engineering degree from the Instituto Tecnológico y de Estudios Superiores de Monterrey, Guadalajara Campus in May 2018. He was accepted in the graduate program in Computer Sciences in June 2018.

This document was typed in using $\LaTeX\,2_{\varepsilon}$[1] by Arturo Silva Gálvez.

[1]The template `MCCi-DCC-Thesis.cls` used to set up this document was prepared by the Research Group with Strategic Focus in Intelligent Systems of Tecnológico de Monterrey, Monterrey Campus.