An environment for a programming practical work based on gamification

January, 2016

**Author:** Àlex Moré Guardiola

**Supervisor:** Jordi Petit Silvestre

**Cosupervisor:** Enric Rodríguez Carbonell

**Department:** Computer Science

*Universitat Politècnica de Catalunya*
Abstract

Gamification is a relatively new concept that every day is more frequent in our society. Gamification is a learning method through game mechanics in non-entertainment applications and environments. The objective is to stimulate the motivation, the concentration or the effort, among other positives values in common with games while learning new skills in the area where it is applied.

In the Barcelona School of Informatics, this method has been used in Data Structures and Algorithms course for some years. Each semester, a practical work takes place as a competition with the aim to motivate the students to solve the proposed problem and help them to consolidate the concepts studied during the course.

This project consists of creating a new environment for this practical. To reach it, we have designed a new game that simulates the its rules and acts as a judge. Then we have created a viewer to watch the games played in a web browser, and finally, we have created a map editor to design new boards to play the game.

The project has been integrated into the Jutge.org server, the virtual learning environment used in the Barcelona School of Informatics with the objective to be released the next semester.
**Resum**

La gamificació és un concepte relativament nou que cada dia és més utilitzat en la nostra societat. La gamificació és un mètode d’aprenentatge a través de mecàniques de joc en entorns i aplicacions no lúdiques. L’objectiu és incentivar la motivació, la concentració o l’esforç, entre d’altres valors positius comuns en els jocs, mentre s’aprenen nous coneixements de l’àmbit on s’estigui aplicant.

A la Facultat d’Informàtica de Barcelona ja fa uns anys que s’utilitza aquest mètode a l’assignatura d’Estructures de dades i Algorismes. Cada quadrimestre es realitza una pràctica en forma de competició amb l’objectiu de motivar als estudiants a solucionar el problema plantejat i ajudar-los a assimilar millor els conceptes treballats durant el curs.

Aquest projecte consta de la creació d’un nou entorn per aquesta pràctica. Per tal d’aconseguir aquest nou entorn, hem desenvolupat un nou joc que simula les seves normes i actua com a jutge. Posteriorment varem crear un visor per poder veure les partides jugades en un navegador. L’últim punt realitzat va ser un editor de mapes per tal de poder dissenyar nous taulers per jugar al joc.

El projecte ha estat integrat al servidor del Jutge.org, l’entorn d’aprenentatge virtual que s’utilitza a la Facultat d’informàtica de Barcelona amb l’objectiu de ser llençat al següent quadrimestre.
Resumen

La gamificación es un concepto relativamente nuevo, cada día más usado en nuestra sociedad. La gamificación es un método de aprendizaje a través de mecánicas de juegos en entornos y aplicaciones no lúdicas. El objetivo es incentivar la motivación, la concentración o el esfuerzo, entre otros valores positivos comunes en los juegos, mientras se aprenden nuevos conocimientos en el ámbito dónde se esté aplicando.

En la Facultad de Informática de Barcelona ya hace unos años que se utiliza este método en la asignatura de Estructura de datos y Algoritmos. Cada cuatrimestre se hace una práctica en forma de competición con el objetivo de motivar a los estudiantes a solucionar el problema planteado y ayudarles a asimilar mejor los conceptos trabajados durante el curso.

Este proyecto consta de la creación de un nuevo entorno para esta práctica. Para lograr este nuevo entorno, hemos desarrollado un nuevo juego que simula sus normas y actúa como juez. Posteriormente creamos un visor para poder ver las partidas jugadas en un navegador web. El último punto realizado fue un editor de mapas para poder diseñar nuevos tableros para jugar al juego.

El proyecto ha sido integrado al servidor del Jutge.org, el entorno de aprendizaje virtual que se usa en la Facultad de informática de Barcelona con el objetivo de ser lanzado el próximo cuatrimestre.
Acknowledgements

First, I would like to thank my supervisors, Jordi Petit and Enric Rodríguez. They have been involved since the first moment we met, and I am so grateful with their attitude in the project. I have never felt I was alone with the work, and this gave me more motivation to carry on with the project.

Secondly, I would specially express my gratitude to my brother Roger, for helping me with the design of the images of the whole project.

Finally, I would also like to thank Anna and my parents for their support during these months. Without them, this project would not have been possible.
Contents

Abstract ................................................................. 3
Resum ................................................................. 4
Resumen ............................................................... 5

1 Introduction .......................................................... 13
  1.1 Context .......................................................... 13
  1.2 Stakeholders .................................................... 15
  1.3 State of the art .................................................. 16
  1.4 Motivation ....................................................... 19
  1.5 Project Outline ................................................ 19

2 Project Management ................................................ 20
  2.1 Objectives ....................................................... 20
  2.2 Scope .......................................................... 21
  2.3 Methodology and rigor ......................................... 22
  2.4 Planning ........................................................ 23
    2.4.1 Dependencies ............................................. 23
    2.4.2 Before the game .......................................... 24
    2.4.3 Runner .................................................... 24
    2.4.4 Viewer .................................................... 25
    2.4.5 Map Editor ............................................... 25
    2.4.6 Deviations ............................................... 25
  2.5 Resources ....................................................... 27
    2.5.1 Material .................................................. 27
    2.5.2 Human ..................................................... 27
  2.6 Costs .......................................................... 27
    2.6.1 Direct Costs .............................................. 28
    2.6.2 Indirect Costs .......................................... 30
    2.6.3 Total Costs .............................................. 31
2.7 Sustainability Report
  2.7.1 Economic Dimension
  2.7.2 Social Dimension
  2.7.3 Environmental Dimension
  2.7.4 Total Score

3 Game Implementation
  3.1 Architecture
    3.1.1 Game and SecGame classes
    3.1.2 Player and Registry classes
    3.1.3 Board class
    3.1.4 PosDir class
    3.1.5 Action class
  3.2 New Features
    3.2.1 Hide opponents
    3.2.2 Ghost Walls
    3.2.3 Poquémon’s attributes

4 The Viewer
  4.1 Architecture
  4.2 Improving the Viewer
  4.3 Parsing file

5 The Map Editor
  5.1 Motivation
  5.2 Architecture
  5.3 Check Balancing Algorithm

6 Conclusions

7 Future Work

A Dependencies

B Gantt Diagrams

C Game Documentation
  C.1 Rules
C.2 Viewer ................................................................. 70  
C.3 Map Editor ........................................................... 71  
C.4 Programming ......................................................... 73  
  C.4.1 Running your first match .................................... 73  
  C.4.2 Adding your player .......................................... 74  
  C.4.3 Playing against the Dummy player ....................... 74  
  C.4.4 Restrictions when submitting your player ............. 75  
C.5 Tips ................................................................. 76  

Bibliography ............................................................ 78
# List of Figures

1.1 *Jutge.org* submission's result screenshot. Hello world problem. . . . . 15  
1.2 *Poquémon’s* board game. ....................................................... 16  
1.3 *Bola de Drac*’s viewer screenshot. .......................................... 18  

3.1 Game architecture ................................................................. 36  
3.2 How does a game work? ........................................................... 36  
3.3 Player’s header class .............................................................. 38  
3.4 Examples of Board class public functions .................................... 38  
3.5 Hide Opponent function ......................................................... 41  
3.6 Ghost Wall data type. .............................................................. 42  
3.7 Ghost Wall main functions ...................................................... 42  

4.1 Viewer’s screenshot ................................................................. 46  
4.2 Parsing file parameters. ......................................................... 47  
4.3 Board parsing. ....................................................................... 48  
4.4 *Poquémon*, Walls and Bonuses parsing. ..................................... 49  
4.5 Actions parsing. ................................................................. 49  

5.1 Map Editor screenshot ............................................................ 52  
5.2 Balancing Algorithm: Recursive call ......................................... 53  

A.1 Before the game dependencies. ................................................. 58  
A.2 Runner dependencies. ............................................................. 59  
A.3 Viewer dependencies. ............................................................. 60  
A.4 Map Editor dependencies. ....................................................... 60  

B.1 The planning of Before the Game phase. .................................... 61  
B.2 The planning of the Runner phase. ............................................ 62  
B.3 The planning of the Viewer phase. ............................................ 63  
B.4 The planning of the Map Editor phase. ...................................... 63
List of Tables

2.1 Project phases. .................................................. 23
2.2 Project phases dependencies. ................................ 24
2.3 Human resources costs. ....................................... 29
2.4 Hardware resources costs. .................................... 29
2.5 Software resources costs. ..................................... 30
2.6 Office costs. .................................................... 30
2.7 Total costs. ..................................................... 31
2.8 Sustainability matrix. ........................................ 34
Chapter 1

Introduction

1.1 Context

Some years ago, professors of Algorithmics of the Barcelona School of Informatics (FIB) at Universitat Politècnica de Catalunya (UPC) noticed many students were not motivated in class nor the practice but they were really interested in computer games. Therefore, they decided to introduce a kind of computer game activity into the course to motivate students in class. Professors of Data structures and algorithms (EDA) [1] course, proposed to students in these last semesters to develop a programming assignment that consists to create a player of a computer game. In order to work out this assignment, the students need to use the knowledge learned in class before. Then they will take part in a competition to know who is the best player [2, 3]. This kind of competition is inspired in the International Collegiate Programming Contest (ICPC) Challenges [4] organized by the Association for Computing Machinery (ACM) during the World Finals of their annual ACM International Collegiate Programming Contest.

The way to perform this competition is playing a game between 4 players and pass rounds whilst your schoolmates are eliminated. The worst player in each round is kicked out of the tournament. This part of the course, the competition, is one of the best experiences of the Degree for most of the FIB students, and this project wants to be part of this experience for current and future students.

It is important to highlight that these games are not the typical interactive video game such as Super Mario Bros or Call of Duty, for example. In these games, students
do not directly play the game. Instead, they have to code a program implementing the artificial intelligence of the player. Each student follows his own strategy and the main objective is to beat his schoolmates and reach the finals of the competition. These programs should be able to react to anything that happens in the game because human intervention is not possible.

These games are stored and handled in Jutge.org [5] server, which was released by Jordi Petit and Salvador Roura in 2009. Today it is the primary on-line tool used in FIB school to evaluate academic programs created by students in programming subjects. The Jutge.org server has currently more than 30 compilers, and you can program in several languages such as C, C++, Python, Haskell or Java. There are hundreds of available problems organized into courses. For example, we can find a course offered by Programming 1, Data Structures and Algorithms or Programming Languages subjects.

This server has several virtual machines that execute the programs in a safe environment. This process starts when a user submits his program. It is pushed to the queue on the virtual machine and then the server returns a feedback with the corrected results (figure 1.1).

In a similar way, the user’s submissions of this assignment will be tested with some executions. Each execution will simulate a match of the game, and the student’s program should win in all these tests the opponent we provide. This player is called Dummy and beating it is required to take part in the competition.

This project proposes the development of a new game, called Poquémon. This game is based on the famous Japanese video game saga Pokémon [6]. It is a kind of RPG (Role-playing game) [7] where each character evolves and changes its attributes, such as the attack power or the defense shield, depending on its actions and the bonuses it collects. This kind of project is divided into two main parts: the Runner (the program that simulates the rules of the game and provides the API to develop new players) and the Viewer (the tool that allows us to watch the matches in a web browser).
Moreover, this project introduces a **new element** unavailable in previous games. This element is a tool called *Map Editor*. It will allow students to develop their mazes for the game and help them to improve their algorithm. A maze is a playable stage of the game, and each match is played in a single maze. Image 1.2 we provide a board example of what *Poquémon* will look like and a game example in the following link: https://www.jutge.org/gameviewer/G00009_en/Viewer/viewer.html?game=sample.

1.2 Stakeholders

The following list shows and explains the different stakeholders of this project:

**Project developer**: This project has one single developer. He is the first interested to finish this project and offer a good product. The responsibility of the project developer is to write the game program (the runner), do the viewer to see the matches, create the map editor and write the documentation of the
An environment for a programming practical work based on gamification

whole project. The developer has to test each part of the project and check there are not bugs in the game.

Project supervisor: This role is shared between Jordi and Enric, the asvisors of this project. Their role is to supervise the project evolution and guide the project developer. A periodic communication between the supervisors and the developer is key to guarantee the success of the work.

Users: In this case the students of EDA perform this role. Both subjects have a similar content, and they will compete together in the same tournament. These students are the most affected part of this project, but they don’t know it yet. Using the API offered by the Runner, and viewing their matches with the Viewer, their role is to design and implement an algorithm able to get more score than their rivals.

Professors: With this project, the professors of EDA and ALG save up to make a new game. They will only have to evaluate the students.

1.3 State of the art

Outside our university, there exist lots of artificial intelligence games as the one we are implementing. We give some examples below:

Angry Birds AI Competition [8]: The goal is to build an intelligent Angry Bird agent who can play better than human players. It is an annual competition located in Australia since 2012.
AI Challenge [9]: An international artificial intelligence programming competition started by the University of Waterloo. A total of four editions were celebrated between 2009-2011.

Battlecode [10]: Programming competition that combines AI, software engineering and strategy. The winner is rewarded with $50,000. It has an annual tournament since 2001.

All these tournaments are similar to the games existing in our university; which are Bola de Drac, PacMan, Apocalypse Now and Tron 3D.

In UPC, this kind of game started in 2007 when the Professor Omer Giménez did the first game in this field called Dominator [11]. This game was introduced in a subject for the Degree in Mathematics. The process was similar to the one we use today. It was in 2010 when this idea was imported to Computer Science too. Since Pr. Giménez left the university, Jordi Petit is the responsible for the maintenance of these games. Until 2011, the website to visualize the games had not changed much from Gimenez’s version.

At this time, the runner and the viewer that displays the games were separated from each other. This new architecture allowed to write a new viewer in HTML language to watch the games in a web browser.

In 2012, Professor Enric Rodríguez took part into the project and created the game Bola de Drac. On the other hand, Jordi Petit with student Mario G. Munzón developed the PacMan game, both with this new format. In 2013, student Albert Vaca did Tron 3D and improved the code of the game ApocalypseNow, as a final Degree project similar to the current one [12]. Rodríguez did a new viewer with javascript for ApocalypseNow game. But nowadays, many projects are needed to avoid always reusing the same games. When a game is reused, it is necessary to think enough new features and change everything possible to make the new game so different that new students could not copy the assignments of old students. Teachers detected plagiarism between assignments of different games and they want to avoid it by developing a new project as different as possible.
Once we have seen the history of the previous games, we must look the common characteristics between all of them. There are two main concepts I would like to emphasize:

**Players:** Each student has to design the intelligence of his character/s. In the first release of Poquémon all the players will have to design the intelligence of only one agent. We thought about a future version with more than one character per player.

**Maps:** The structure of the maps is similar in the current games. All of them have walls and bonuses distributed in the board. Currently a map has a fixed wall distribution and do not change never. In Poquémon, this idea will change. This new game will introduce dynamic maps to create more exciting games.

So, we can take Bola de Drac as a reference for Poquémon because they have a similar game structure. Both games use the same actions: move the player to an adjacent position on the board or attack an opponent player. Poquémon’s agents have attributes (attack, defense, and scope) that can be improved, but they do not have states as in Bola de Drac. There, in Bola de Drac, the agents change their state depending the bonus they take, but the attributes do not change at any moment. In the figure 1.3 we can see a screenshot of Bola de Drac, and it gave us an idea of what Poquémon would be.

![Figure 1.3: Bola de Drac’s viewer screenshot.](image-url)
In conclusion, the main architecture of the game *Bola de Drac* could be reused, but the most of the code was replaced with *Poquémon*’s game code. It is crucial to implement a reusable and understandable code to help the development of future versions of *Poquémon* or some possible new games based on it.

### 1.4 Motivation

The chance of creating a programming practice for students is not always present. This project gave me the opportunity I did not have when I studied the 2nd course of the degree and I did a similar practice but with the student role.

Designing a new environment with a new topic, rules, features and everything you can imagine is a kind of project I had dreamed of doing in the career, and I never had the chance to do. So, I did not hesitate to develop this project when I heard about it.

The latter motivation I found with this project is that I had the chance to help the students to consolidate the concepts they have studied in the course, and I wish they will enjoy doing this programming practice as I did some years ago. This reason is the most important in the choice of the final degree project.

### 1.5 Project Outline

This document is organized as follows:

**Chapter 2**: Provides the project management information such as the objectives, the temporal planning, the costs of the project or the sustainability report.

**Chapter 3**: Explains the architecture of the game including the rules and the players designed for the development of the project.

**Chapter 4**: Illustrates the Viewer details and a brief summary of the content.

**Chapter 5**: Describes the Map Editor and exposes the origin of the idea, the resources used and shows the way to link this tool with the game.
Chapter 2

Project Management

This chapter explains the concepts related to the project management. We provide the objectives of the project, the planning followed, a budget estimation and a sustainability analysis as main topics.

2.1 Objectives

The first objective of this project is to create a new game for the practical programming of EDA’s subject. To give more value to this work, we thought to develop a Map Editor, currently nonexistent. Students will be able to design and create their boards for the practice, and some of them will be introduced in the competition we will do the next course. Finally, we find educational goals in this project such as introducing didactic materials to students and encouraging them to develop their program. We aim to give them an effective learning of programming, data structures, and algorithms and help them to consolidate the knowledge they have learned during the course. Therefore, the goals of this project can be summarized in the following points:

Create a new environment: The new environment has to include features currently unavailable. The topic of the project must be familiar with the students to give them more motivation to develop the practice. And, finally, the rules of the game should be easy to understand for a best accepting between the students.

Develop a map editor: A map editor is a tool unavailable in previous games. In this project, we want to introduce a new one, mainly for the students and
teachers. This tool will involve the players to design their maps to improve the quality of their algorithms.

2.2 Scope

This project can be divided into different parts to understand it more easily. First of all, we have the Runner, which contains the rules. The content of this section makes possible the game to works. Next, we have the viewer, which is the interface users will use to watch the matches. Finally, we find the map editor, that is the part where users can use their imagination to create their game boards. The project will have its documentation and students will have available the game rules and instructions of the Map Editor and the Viewer.

In the following items we will show in more detail the scope of each part of the project:

- **Runner**: This part is divided into two important blocks. The first one is the **skeleton of the game**. The objective of this part is to create a game with no mistakes and with the capacity of being reused in the future with new rules and new properties. The initial version of this game will be adapted to play with four players and with one agent for each one. The second part of the runner is the **different players** we will implement for the students. The most important will be the *Dummy* player, who will be the opponent of the students. The second player will be the *Demo* player, just an example for the students to help them to begin with the practice. The last player will be the *Smart* player, used to calibrate the *Dummy* and it won’t be available for the students.

- **Viewer**: The viewer has a simple goal; be able to play the matches played before. When a game is loaded in the runner, the data file with the full game information is read. The viewer will reproduce a video that will be as a time-lapse (at least one image per round). Each match is played in a maze. Some of the maze’s walls appear or disappears every $N$ rounds, and that makes more dynamic the game.

- **Map’s editor**: This is the newest part of this project. It has no precedents in previous games. So far, each map needed to be written directly by editing
a text file, without any aid. This kind of input is error-prone and makes hard and slow the task of creating a new map. To avoid all these problems, we have the purpose and principal goal with this tool to let the users create their boards with an easily and fast way for the competition.

To create this tool, we will associate each parameter of the map to a variable and the user will be able to control each value. We won’t let to input invalid values (negatives, for example). When the user exports the maze, we will show an alert if there are any mistake in the board. That will help to avoid future problems in games with this map.

2.3 Methodology and rigor

This project is implemented in C++ (the Runner part), JavaScript and HTML (the Viewer and Map Editor part) programming languages. We used Eclipse software to develop it and the version control chosen is Bitbucket, to have a private repository and hide the code.

The methodology chosen to carry out this project is Kanban [13]. This method was created by Toyota in 1962 and consists of a set of cards where each one indicates what we have to do in the project and when. These cards help to classify the priority of each task. As regards the proposed project, Kanban has the following characteristics:

1. A simple and understandable process.
2. Provides quick and precise information.
3. Provides quick response to changes.
4. Minimizes the waste of time in our project.
5. Control can be maintained.
6. Delegates responsibility to line workers, in this project just the developer.

To apply this methodology we wrote on a post-its all the remaining tasks and we classified them as to do, doing or done.[14] One of the important ideas is when the developer starts a task; he cannot change to another until he finishes it. This methodology is helpful to see an overall state of the project, and it makes general
changes easier to do. There are also some other classifications as put the post-its on a calendar to indicate when the developer has to do the work. This methodology is the same as you can find in Trello’s application, available for Android and IOs. We used it too to have always the updated state of this project readily available.

To have a solid monitoring of the project, each post-it had its deadline to avoid be late in the delivery deadline agreed with the coordinators. To be sure that everything is alright, the game was tested and debugged frequently. If some error appeared, we knew this particular part of the project was not finished. When we were able to run several executions with no mistakes we knew this part was done and then we can move the task from doing column to done column.

2.4 Planning

In this section, we will explain the temporal planning of the project we did at the beginning and we will comment the dependencies it has. The goal of this plan was to finish the project within the given time. Each Gantt’s diagrams done for this planning are attached at the end of the document, in Appendix B.

2.4.1 Dependencies

This project was divided into four general phases following its structure, as we can see in the table 2.1. Each step will be explained later with more details. On the other side, the project has a lot of dependencies between each phase. Each one we have in the table 2.1 needs to wait for the previous stage before we can start it. So in the table 2.2 we have the principal table dependency of the project.

<table>
<thead>
<tr>
<th>#</th>
<th>Phase</th>
<th>Days</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Before the game</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>Runner</td>
<td>62</td>
</tr>
<tr>
<td>3</td>
<td>Viewer</td>
<td>35</td>
</tr>
<tr>
<td>4</td>
<td>Map Editor</td>
<td>37</td>
</tr>
</tbody>
</table>

Table 2.1: Project phases.
An environment for a programming practical work based on gamification

<table>
<thead>
<tr>
<th>#Phase</th>
<th>#Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 2.2: Project phases dependencies.

Now we will see with more details the planning and the goals of each phase. For each phase, we have done a Gantt diagram where we can see the approximated duration of each one. The Appendix A provides the detailed dependencies of the tasks for each phase. The resources and the potential deviations for each phase are explained in the next section.

### 2.4.2 Before the game

This first stage of the project was not a physical part. It included the time spent on thinking about the idea of the project, the topic and the rules of the game. This period started in the last week of June when Jordi, Enric and I met to agree with the development of this project. Before carried on with the project, my supervisors approved the project idea I showed them. In the figure B.1 (in the Appendix chapter) we find the planning for this initial part and we can see that is unable to parallelize any subtask and each one (except the first one) depends on the previous task. The goal of this first stage is to set the rules of the game.

### 2.4.3 Runner

The runner was the first component we must develop in this project once we have established the rules of the game. The main goal of this phase is to develop the game, test it and prove it works. This stage accepts some tasks in parallel as we can see in figure B.2. The figure also includes empty days, that I had not my computer available.

The first we have to do is program the runner classes, mainly the board class. It will be the responsible for update the game state each round taking care that no rule is violated. When we finish this task, we can create the players we need. In this case, we will create the Smart player and then, taking as a template this one, we will
develop the *Dummy* player explained before. Now we can start to develop the viewer.

### 2.4.4 Viewer

The second section we had to implement is the Viewer tool, that will allow us to watch the matches in a web browser. We can see in the figure B.3 an approximation of the process we have to follow. Before start the Viewer we had to be sure that the runner worked with no mistakes. The Viewer will help us to see any bug we were unable to identify while we were testing the Runner. The tasks of the Viewer were not as detailed as in the Runner section because we did not know exactly all the steps. We knew we had to start by writing the file that will update the state of the game each round. This file will be used by the HTML file that we also had to implement and then we will be able to watch a full game.

### 2.4.5 Map Editor

The Map Editor is the newest part of the project. This section was developed the last one because it needs the Runner and the Viewer finished to be useful. Without the them, this part of the project makes no sense. The objective of this phase is to develop a tool that will allow users to create their mazes for the game.

This planning was created following the Viewer’s tasks as a reference to estimate as closest as possible to the reality how many and which tasks would have the development of the Map Editor. We did it because the architecture of both tools are similar. All the tasks included in this section has a precedence on his previous task as the Viewer phase. In the figure B.4 we can see the detailed tasks of the Editor we planned.

### 2.4.6 Deviations

During the development of the project, we have not suffered many deviations if we look just the tasks planned and ignore the temporal distribution. We have followed each phase, and we did minor changes respect the tasks we provide in this document. These modifications are mainly the amount of meetings with the supervisors that were not accurate in the plan. The Runner caused a few delay too because there was
always details to improve or change that made spent more days than expected.

The second point that generated a delay was because we did not include the project management in the plan and it took more time than we thought initially to develop a good work and analysis. This mistake forced us to extend the development of the Viewer, and that implied a minor delay with the last phase. We started the Map Editor with some weeks of delay but, as we explained before, we left the lasts weeks of the plan with no tasks to have this error margin and avoid to run the lasts days.

If we did not have this delay, we could add more features and details to the game as the ones we propose in this document.

So, in conclusion, the planning provided had no deviations if we look the tasks but it is true we have few delay in the temporal plan that, fortunately, has not produced any consequences to the final result.
2.5 Resources

In this section, we will explain the material and resources needed to develop this project and the changes we did if some problem appears.

2.5.1 Material

To do this project we need a computer able to compile a C++ project as well as able to run JavaScript files. It is important to have a web browser installed to watch the matches and to create the mazes with the Map Editor. The last resource we need is the server and the web domain to upload the game to Internet and run it on the server.

There should not exist any problem that make me change the resources needed. The only way to develop this project is using the hardware we have explained before. The computer is needed in all the phases, but the server is only required when we finish the last phase.

2.5.2 Human

This project has three people involved. There are Jordi and Enric, the supervisors who have the responsibility of approving or rejecting my suggestions about the game and guide me in the project. The third person is me, the developer. We need each member in each phase: the supervisors have always to verify and accept the proposals I give them.

2.6 Costs

In this section, we will estimate the cost of the project development considering the different tasks we planned and the resources needed. We will simulate we are a company located in an office in Barcelona to estimate the total cost. We use more than a computer, and the roles that take part in the project are the supervisor (only one), the developer, the tester, and the graphic designer. The director and the developer are required in each phase. The tester is required in each phase except the before the game stage. And the graphic designer is needed only in the Viewer and
the Map Editor phases to create the images of the tools. To simulate the budget, we contemplate direct and indirect costs.

2.6.1 Direct Costs

These costs are related to the tasks described in Gantt’s diagram (Appendix B). They are computed following the resources and time needed to develop the project. We have considered four daily hours in a workday to do the budget.

Human resources

To develop this project we need the following roles:

1. **Project Manager**: The supervisor of the project. He/She must approve each task developed and be ensure the project guarantees the planning and the budget agreed. The project manager only works on the project when meets with the developer and helps to solve any problem.

2. **Software Developer**: Responsible of the development of the project and have constant communication with the project manager and with the software tester to solve the bugs.

3. **Graphic Designer**: His task is to design and create the images of the Viewer and the Map Editor.

4. **Software Tester**: Responsible of execute the program several times and search the existing bugs. He must communicate immediately to the software developer the mistakes found.

The workers salary have been taken from trovit’s website.

Hardware

The hardware resources we need to do this project is a computer for each worker. Programmer’s and tester’s computer is a *HP Pavilion 15 n216sf* and the graphic designer’s computer is an *iMAC 27”*. We also need a server and a web domain to save the program and run the matches. Finally, we have considered the office’s
An environment for a programming practical work based on gamification

<table>
<thead>
<tr>
<th>Role</th>
<th>#Days</th>
<th>Price/Hour</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Manager</td>
<td>7</td>
<td>22 €</td>
<td>616 €</td>
</tr>
<tr>
<td>Software Developer</td>
<td>128</td>
<td>10 €</td>
<td>5.120 €</td>
</tr>
<tr>
<td>Graphic Designer</td>
<td>7</td>
<td>12.5 €</td>
<td>350 €</td>
</tr>
<tr>
<td>Software Tester</td>
<td>30</td>
<td>8 €</td>
<td>640 €</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>6.726 €</strong></td>
</tr>
</tbody>
</table>

Table 2.3: Human resources costs.

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
<th>Useful Life (years)</th>
<th>Usage (days)</th>
<th>Depreciation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmer’s Computer</td>
<td>800 €</td>
<td>4</td>
<td>128</td>
<td>71 €</td>
</tr>
<tr>
<td>Graphic Designer’s Computer</td>
<td>2.129 €</td>
<td>4</td>
<td>7</td>
<td>11 €</td>
</tr>
<tr>
<td>Tester’s Computer</td>
<td>800 €</td>
<td>4</td>
<td>30</td>
<td>17 €</td>
</tr>
<tr>
<td>Server and Web Domain</td>
<td>3.000 €</td>
<td>4</td>
<td>unknown</td>
<td>3.000 €</td>
</tr>
<tr>
<td>Office’s Furniture</td>
<td>3.600 €</td>
<td>8</td>
<td>160</td>
<td>198 €</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>3.297 €</strong></td>
</tr>
</tbody>
</table>

Table 2.4: Hardware resources costs.

furniture in the budget because we need it to develop the project in the office. In the table 2.4 we can see the depreciation cost of each resource.

The Programmer’s and tester’s computer price was found in the Fnac website. The graphic designer’s computer has the official price we can find in apple’s website.
Software

To do this project, we have used several open-source software. In the table 2.5 we can see them.

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubuntu 14.04</td>
<td>0 €</td>
</tr>
<tr>
<td>Eclipse Indigo</td>
<td>0 €</td>
</tr>
<tr>
<td>Web Browser</td>
<td>0 €</td>
</tr>
<tr>
<td>Bitbucket Repository</td>
<td>0 €</td>
</tr>
<tr>
<td>LATEX</td>
<td>0 €</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>0 €</strong></td>
</tr>
</tbody>
</table>

Table 2.5: *Software resources costs.*

### 2.6.2 Indirect Costs

The indirect costs of this project are directly related to the office expenses, and there is also the Internet connection. In the table 2.6 we can see these costs.

<table>
<thead>
<tr>
<th>Product</th>
<th>Price/month</th>
<th>Usage (months)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Office’s Rent</td>
<td>1.199 €</td>
<td>5</td>
<td>5.995 €</td>
</tr>
<tr>
<td>Internet Connection</td>
<td>38.60 €</td>
<td>5</td>
<td>198 €</td>
</tr>
<tr>
<td>Water, electricity and air conditioning</td>
<td>423 €</td>
<td>5</td>
<td>2.116 €</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>8.309 €</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.6: *Office costs.*

The price of the office’s rent is from *habitaclia*’s website. The Internet connection is the price that *Movistar* offers to companies in his web and the water, electricity and air conditioning cost is an extrapolation of the data found in *asociacion3e*’s website.
2.6.3 Total Costs

We have considered a 10% contingency for possible delays or problems. The project tasks are detailed and should not appear any problem. All the prices include the taxes we have in Spain, 21% IVA. In the table 2.7 we can see the total costs of the project.

<table>
<thead>
<tr>
<th>Type</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human Resources</td>
<td>6.726 €</td>
</tr>
<tr>
<td>Hardware Resources</td>
<td>3.297 €</td>
</tr>
<tr>
<td>Software Resources</td>
<td>0 €</td>
</tr>
<tr>
<td>Total Direct Costs</td>
<td>10.023 €</td>
</tr>
<tr>
<td>Office Costs</td>
<td>8.309 €</td>
</tr>
<tr>
<td>Total Indirect Costs</td>
<td>8.309 €</td>
</tr>
<tr>
<td>Contingency</td>
<td>1.833 €</td>
</tr>
<tr>
<td>Total Costs</td>
<td>20.165 €</td>
</tr>
</tbody>
</table>

Table 2.7: Total costs.

The cost of each stage of the project can be calculated following the equation 2.1.

\[
\frac{totalProjectCost}{totalDurationInDays} \times durationPartInDays = costPart \quad (2.1)
\]

So, if we compute the cost of each part using the equation 2.1 we find that the before the game part costs 1.718 €, the runner part costs 9.968 €, the viewer part costs 3.208 € and the map editor costs 3.438 €. The duration part of each phase has been extracted from the Grantt’s diagram we did.
2.7 Sustainability Report

In this section, we will do the sustainability evaluation of our project. We will focus on economic, social and environmental dimensions. Finally, we will award each dimension with a punctuation between 1 to 10 depending the global valuation of each one.

2.7.1 Economic Dimension

Previous sections show the costs related to the project. These costs include both material and human resources to carry out the project but it does not contemplate the cost of future updates or the maintenance. So, if someone wants to do a new update version in the future, he or she should do a new budget with his goals with the new version of the project.

The cost of this project is certainly viable since more than half of the budget is intended for pay the office’s expenses and the possible contingency of the project. That means the project is not expensive and if we would replicate a cheap project we could only replicate it if we work faster than this time. This would imply few expenses of indirect costs because we would work for fewer months. And maybe we would be able to save human resources just for working fewer hours. That would mean a result project of a lesser quality than expected. On the other side, we cannot reduce the company stuff, we are the minimum possible people working on this project. The resources implied in the project are also essential, and we cannot discard any of them.

The punctuation gave in this dimension in table 2.8 is based on human resources needed, time invested on the project and the quantity of work done in this project.

2.7.2 Social Dimension

This project is developed to be used in an academic context, just for UPC University. Our targets are the students, who are interested in this game to develop their practice the best they could, and learn as much as possible. There are another group benefited with this project; they are the professors of the subjects, who they save up of developing a new game for the next course.
We are developing the project in Barcelona, one of the leading cities in the world in technology and this business can be a perfect bridge to have more projects in the future in our company. This project has a real need behind it. Without it, any student should not be able to develop his practice.

We also consider that any collective can be harmed with the development of this project. Each collective involved in this project need it to make easier his obligation: students will learn playing, and professors will have an easy tool to evaluate his students.

To set a punctuation in this dimension, we have evaluated the volume of people involved and the social impact this project can have.

### 2.7.3 Environmental Dimension

The resources needed to develop this project are mentioned in costs section. Each stage of the process requires a computer to finish it, and when the project will be finished, we will need to have the server running always. So, the project’s resources that could affect the environmental sustainability are the computers and the server. They consume electricity, and that is the main cause.

Another resource we can add to this group is the office’s energy expense, mainly water, electricity and air conditioning. We have done an estimation of the total electric consume of the office. We spent 2143 kW each month; that means we generate 750 kg CO₂ approximately. When finished the project, just the server would produce CO₂ and that is a few quantity.

There are a lot of informatics projects with poor ecological footprint, and this project is not an exception. The positive side is that this project does not have a great environmental impact and we cannot consider it so bad as it would be in a larger context.

The level of energy used in this project cannot be much reduced. We are using the minimum possible resources to finish the project, so it is hard to decrease the pollution of the project.
Some parts of old games can be recycled but anything more. That let us save time, resources and generate less pollution comparing the quantity we would emit if we started the project from zero.

The punctuation of this dimension would be evaluated as the first two. We will considerate the global analysis of the environment dimension and we will set a punctuation.

2.7.4 Total Score

In the table 2.8 we provide the total score of the sustainability matrix based on the analysis done in this section.

<table>
<thead>
<tr>
<th>Sustainable?</th>
<th>Economic</th>
<th>Social</th>
<th>Environmental</th>
<th>Total Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>25</td>
</tr>
<tr>
<td>Results</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>25</td>
</tr>
<tr>
<td>Risks</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total cols.</td>
<td>16</td>
<td>18</td>
<td>16</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 2.8: Sustainability matrix.
Chapter 3

Game Implementation

In this chapter, we will deep in the implementation of the game to understand it. The new features included in this environment will be explained, as well as the architecture of the Runner. We strongly suggest to read the Appendix C.1 for more details about the rules of the game. This attached document is the documentation the students will receive when they develop the programming practice.

3.1 Architecture

When this program is executed, there are some tasks that repeat the same actions one and another time. Instantiate the players or create a board are examples of these tasks. The code that do each part of these tasks can be reused from previous games and we have to adapt the needed parts to our environment. When the game starts, the program uses another different classes. Therefore, these classes are the ones we have completely modified. As we said before, the programming language used in this part of the project is C++.
Before deep with the details of the game, we want to provide an overview of the architecture of the whole project and the execution of a game. In the image 3.1 we can see the global performance of the project and, to provide more details about the runner part, we find the image 3.2 where we can see the tasks done in a single game. In the next subsections, we will deep in each class of the diagram and the functionality in the program. The classes that start with AI are the players of the match and we won’t have the code access in a casual game.
3.1.1 Game and SecGame classes

These classes are the starting point of the execution. If the user uses the SecGame, the program is run in the server. That provides a safe execution environment. The Game class is the one students will use, and its function is to test the student’s players in a local mode.

When the game starts, after instantiating the players, set the game constants and initialise the board, will do the following loop in the Game/SecGame class:

1. Write the board state to the output file. This will allow us to watch the game with the Viewer.
2. Hide the opponents the player cannot see (this is one of the new features that we will explain below)
3. Send a copy of the customized board to each player.
4. Get the actions done by each player. Shuffle them, and apply these actions to the original game board.
5. Print the actions the players asked and did to the output file, for watch the game later.

When we arrive at the last point, we can start again with the next round of the match. When the game finishes, we can see the final score on the terminal screen to know the winner. If we want, we can watch the movie of the game with the Viewer that we will explain in the next chapter.

3.1.2 Player and Registry classes

The player class will be the class extended by the students to develop their programming practice. This class gives access to the Action and Board class and use their functions as we can see in the figure 3.4.

Moreover, we have the Registry class. The main function of this class is to instantiate the players of the game. The problem we find here is the main program does not know how many Players will be inherited from the Player class. So, to avoid
An environment for a programming practical work based on gamification

class Player : public Board, public Action

Figure 3.3: Player’s header class

this problem, each subclass of Player will add itself to the Registry by name. That means we use the Factory pattern to instantiate the player objects.

3.1.3 Board class

The main function of the board class is storing the game state each round. This class provides the read-only functions the players will use to know each available parameter of the game.

Only the public functions can be accessed by the players when they develop their programs. The board state has no way to be modified by the users. In the environment we are developing we provide some functions as we can see in the following code:

```cpp
int nb_rounds();

bool pos_ok(const Pos& p);

CTYPE& cell_type(const Pos& p);
```

Figure 3.4: Examples of Board class public functions

The first function returns the number of rounds of the game. The second validates the given position, and the last function returns the type of the cell placed in the position P. The available types of cells are: Empty, Wall, Stone, Point, Scope, Attack and Defense.

Another important part of the board class is the input and output functions. This class has the task to read the game state each round as we said, but it has to print the next state of the board, the actions asked by the player and the actions finally they have done. The functions that have to print the state are called `print_preamble(ostream)` (that prints the parameters of the game) and `print(ostream)` (that prints the board, the Poquémon data, the Ghost Walls data and the Bonuses data). The constructor
of the class is the responsible of load the game state given the output of the previous round generated by the functions commented.

Finally, we have to explain the way to modify the board and apply the actions the user asked. That is why we have the function \( \text{next}(\text{actions\_asked}, \text{actions\_done}) \). This function is the link between the \textit{Game} (or \textit{SecGame}) and the Board class. The steps we follow to update the game state are the following:

1. Starting from the 4th stage of the \textit{Game} class, we have to shuffle the actions received.

2. For each action, we must check it can be done with no rules violation. If the action is legal, apply it to the board and update the game state. Otherwise, the action requested becomes \textit{undefined} and the player do not do any action.

3. Check the regeneration of the \textit{Ghost Walls} (see section 3.2.3), the Poquémon and the bonuses. If any of them need to be placed again on the board, we search for a random position that do not benefit any user using the function \( \text{get\_random\_safe\_position} \ (\text{type\_to\_regenerate}, \text{current\_board}) \).

4. Return the updated board and the actions finally done.

### 3.1.4 PosDir class

Class that stores the possible directions we can use in the game. These directions are \textit{Top}, \textit{Bottom}, \textit{Left}, \textit{Right} and \textit{None}. They can be used to move a character or attack to any of these directions.

We can do logical operations with two positions (as equal or not, greater or less, for example). The addition is also accepted using the \(+\) operator or \(+=\) to moving one cell to the desired direction.
3.1.5 Action class

This class stores the available actions in the game. In our game, the allowed actions are:

**move**: Move your Poquémon to an adjacent position. This position cannot contain a Wall or another Poquémon. Function used: \textit{move}(Dir \ d)

**attack**: Attack an opponent Poquémon. Both must be aligned at the same row/column without walls between them. Function used: \textit{attack}(\textit{Dir} \ d)

**undefined**: The Poquémon do not do anything and keeps its position until the next round. Any function is required to do this action.

The parameter \textit{Dir} \ d means the direction of the action. Actions will be shuffled at the beginning of the round, and each one will be checked before doing it, as we explained in the last section.

3.2 New Features

In this project, we need to introduce some features in the game to make it different from the rest of the existing environments. The provided points detail these new features and justify why we decide to introduce them in the game.

3.2.1 Hide opponents

This new feature is used in the \textit{Game} and \textit{SecGame} class. Each round the board is customized for each player, and everyone will receive a board where not all the opponents would be located. In the figure 3.5 we can see the algorithm we did to customize the board.
void Board::hide_opponents(int idPlayer) {
  vector<Pos> poquemonPlayer(nb_poquemon());
  for (int i = 0; i < nb_total_poquemon(); ++i)
    if (poquemon(i).player == idPlayer)
      poquemonPlayer.push_back(poquemon(i).pos);
  for (int i = 0; i < rows(); ++i)
    for (int j = 0; j < cols(); ++j) {
      int idCell = cells_[i][j].id;
      if (idCell != -1 and poquemon(idCell).player != idPlayer)
        if (not poquemon_visible(poquemonPlayer, Pos(i,j)))
          cells_[i][j].id = -1;
    }
}

Figure 3.5: *Hide Opponent function*

In the first loop, we get the position of all the Poquémon placed on the board. In the second loop we set to -1 the id of each cell where an opponent Poquémon is placed, and it is not visible.

An opponent is visible when it is located at the same row or column as the processed Poquémon when the player id of both Poquémon is different, and when there are not any wall between them.

The cost of this algorithm is $O(n)$, where $n$ is the number of cells the board has. We do not consider the cost of getting the position of all Poquémon because it is a negligible percent of the $n$ value.

### 3.2.2 Ghost Walls

This is one of the most interesting features of the project. The aim of this idea is to create dynamic mazes that change while the game is playing. This kind of wall has a period property that makes change its state to *present* or *not present*. In the image 3.6 we expose the structure of the *Ghost Walls* used in the game. The attribute *pos* indicates the position in the board. The *present* attribute means the state of the Ghost Wall. Finally, the *time* integer represents the remaining number of rounds to change the state.
\begin{verbatim}
struct GhostWall {
    Pos pos;
    bool present;
    int time;
};
\end{verbatim}

Figure 3.6: Ghost Wall data type.

When a Ghost Wall is present in the board, the occupied cell becomes a Wall cell type. When this Wall changes the state, this cell is transformed to an Empty cell type.

To have a total control of the Ghost Walls in the game, we have created an array with the data of them that updates when each round finishes. These Walls do not change their position during the game, so users can identify them using the functions available in the Board class. In the following image (see 3.7), we provide the header of the functions related with the Ghost Walls.

\begin{verbatim}
int nb_ghost_wall();
int wall_change_time();
int ghost_wall(Pos p);
\end{verbatim}

Figure 3.7: Ghost Wall main functions

The first function returns the number of the total Ghost Walls presents in the board. The second the period (number of rounds) of all the Walls. And the last one returns the rounds remaining to change the state of the Ghost Wall placed in the position P (-1 if it is not a Ghost Wall).
3.2.3 Poquémon’s attributes

The last new feature we want to comment is related with the Poquémon. In this environment, we had not used the same idea as preceded in previous games, the states of the characters. This time, we have introduced attributes in the Poquémon. The main difference between a state or an attribute is that the first give you a temporary benefit (or prejudice) depending the bonus collected, and the second concept, the attributes, give to the character a permanent benefit. In our case, we have bet for the attack, the defense, and the scope attributes.

A second decision related with the attributes is that, in this case, only the scope is a permanent benefit. We have preferred the attack and the defense attributes be modified through the battles. This was the only way to avoid the users get a large advantage in the game.
Chapter 4

The Viewer

In this chapter, we will explain the details of the Viewer. We will also provide an example to understand the relation between the Viewer and the Runner, explained in the previous chapter. Finally, we will describe the improved Viewer and we will deep in the parsing file we use in this tool.

4.1 Architecture

The Viewer is a tool that allows the users to watch in an easily way the games played before and to think how to reach a better score improving their strategy.

This tool is an independent program that takes as input the output of the Runner, seen in the last chapter. Thanks to this independence we can write this tool using another language different from C++. The used languages are Javascript and HTML. Due to this mix of programming languages we used, the games can be displayed in a web browser.

The Viewer is divided mainly into two files:

**HTML file:** This file includes the interface of the Viewer. In other words, this file includes everything we can watch on the screen. Before starting the game, we must select an input file that must contain the game information. This data is the result of the *print* functions called each round of the game in the *Board* class explained before.
When the game is loaded, then start the animation. The board will be drawing in an HTML canvas object. On the top of the screen, we have included some controls to pause the animation or move to a particular round. In each corner of the Viewer, we can see the scoreboard of each player. The colours used for distinguish the players are green, red, blue and yellow.

The scoreboard of each user is updated at the end of each round. If they collect a point bonus or kill an opponent, they will update the points field. The attributes update depending the bonus collected or the action done. When a Poquémon dies, it disappears until the penalisation expires.

**Javascript file:** The second file we coded to develop the Viewer was written in Javascript. The objective of this file is to read the input data and update the board state each round of the game with the information read. This process loops until the game is finished and simulates an animation to the user.

This version of the Viewer includes a new feature that consists that the avatar of the Poquémon flips when it changes its direction from left to right and vice versa.

In C.2 section is available more details about the Viewer. We suggest to read it to know more information about it. In the following image, we have an image that exemplifies everything we have explained in this section.
An environment for a programming practical work based on gamification

Alex Moré Guardiola

Figure 4.1: Viewer’s screenshot

The green Poquémon is dead, and it does not appear in the game because it has not already finished the death penalty. The red player is attacking to the blue player, and finally, the yellow player is going to collect a bonus to increase its attributes. Each bonus is easy to distinguish from others and cannot produce any confusion.

4.2 Improving the Viewer

In the next months, as we said before, we will realise a competition between all the students of Data Structures and Algorithms course. During the tournament, we will provide them a different Viewer respect the version they will see the final day. This day, the best students will be able to watch in live the final rounds and know who is the winner.

The Viewer of this day will include a feature that do not have the initial version; sounds. We have added sounds in the following situations during the game:

When a Poquémon collects a bonus. Each bonus has its sound effect.

When a Poquémon attacks an opponent.
A soundtrack will be playing while the game runs.

A short song will play when the game finishes.

We also provide a button to activate and deactivate the music (the soundtrack) but not the sounds effects.

4.3 Parsing file

The gear that makes work the Viewer is the output of the Runner. The file we receive is always written following the same pattern. In this section, we will deep in this pattern and we will explain what does this file includes that allows us to recreate this animation of the game played. The list below contains as points as sections have the parsing file. That would facilitate the comprehension of it:

1. **Game Parameters**: The first lines includes the parameters of the game. In the image 4.2 we can see some of them. Mainly they are the number of each kind of element present (Poquémon, players, bonus, walls, etc.) and the regeneration time of each one. There also are the maximum stones and scope allowed as well as the size of the board.

   \[
   \begin{align*}
   nb\_players & \quad 4 \\
   nb\_rounds & \quad 400 \\
   nb\_attack & \quad 4 \\
   point\_regen\_time & \quad 50 \\
   battle\_reward & \quad 15 \\
   max\_scope & \quad 8 \\
   rows & \quad 15 
   \end{align*}
   \]

   Figure 4.2: *Parsing file parameters.*

2. **Board**: The second block includes the board of the game. We have used a matrix to simulate it and each position is represented by a character. The available options are: Wall (X), Point bonus (P), Stone bonus (S), Scope bonus (R), Attack bonus (A), Defense bonus (D) and Empty (.). The Poquémon are not represented with a character because we want to hide their position on the board as we saw in the rules. In the following image (4.3) we can see an example of parsing board.
3. **Poquémon:** When we have the board we have to declare all the Poquémon that will play the game. To do this, we have created a kind of table that contains the following data: id, player, row and column placed. Then initial attack, defense and scope the Poquémon had, the number of stones and points it has collected, the time remaining to regenerate hit (if dead, otherwise 0) and the alive state (a or d).

4. **Walls:** The next point to parse are the Gghost Walls placed on the board. We have also created another table with the position (row, column) of this wall, the time remaining to change its state (present, or not) and the state value (1 if present, 0 otherwise).

5. **Bonus:** Now we analyse the bonus of the game. In this case, the table created includes the type of the bonus that is the same letter as we used in the board parsing. Then we have the position (i,j), the points value of the bonus, that will always be zero except for the Point bonus. Finally, the time remaining to regenerate the bonus (if it was collected) and the present attribute that is 1 if present and 0 if not. In the image 4.4 we provide a small example of the three tables commented in this section.
6. **Actions**: The last we must analyse are the actions the Poquémon asked and the actions they finally do. Each requested action is evaluated in the runner and, if it is legal, this action would be done. Otherwise, this Poquémon would do the *undefined* action. In the following image (4.5) we can see an example:

```
actions_asked
player action direction
0 a r
1 m l
2 a l
3 m r
```

```
actions_done
player action direction
0 u n
1 m l
2 a l
3 m r
```

Figure 4.5: *Actions parsing.*

Now we have read the initial state and the first action of the Poquémon, we will loop *nb_rounds* times reading these commented parts and updating the viewer simulating the animation. Is not necessary read the game parameters each round because they do not change during the game. In the following link we give an example of a parsing file: https://www.jutge.org/gameviewer/G00009_en/Viewer/sample.out.
Chapter 5

The Map Editor

In this chapter we will explain the remaining part of the project; the Map Editor. As the preceded parts, we will present the architecture of the tool and then the features it has. We will also provide the details of the checker algorithm we have implemented. This algorithm proves that the board created is balanced and any player has an initial advantage versus its opponents. But firstly we want to do an introduction of this point and explain the origin and the reason about this tool.

5.1 Motivation

The aim of this Map Editor is to encourage the students to create their custom mazes for the competition. This tool was born when we decided to add an important new feature in the practice. This tool is based on the newest Nintendo’s video game, Super Mario Maker [15]. The develop this Editor we take as template the project tinyMapEditor [16], developed by Kodo Games [17].

We decided to develop a Map Editor instead another feature or tool because students have never had the chance to create a board in an easy way, and we want to give them this chance. This tool can be a significant help to improve their strategies and a good tester for any possible game situation.

Our desire is this tool keeps present in the future projects as the Viewer has been.
5.2 Architecture

The architecture we followed to develop this Map Editor was similar to Viewer. We have divided the tool into two layers: the interface layer and the computing layer. The first was written with HTML language and the second with Javascript. The tiles and the images used in the Map Editor were stored in the folder `assig`.

**HTML file:** As we said, this file is the interface of the tool. This time, the page has been divided into three columns. The left column has some submenus; the first includes each element that can be placed on the board. The second shows the tile selected to be placed, and the last submenu contains the principal functions of the map editor. These functions are:

1. **Tile Selector:** Allows to edit the initial configuration of the Ghost Walls.
2. **Tile Eraser:** Allows to delete the content of a cell.
3. **Clear Board:** Remove each element of the current board.
4. **Export Map:** Open a new tab with the code of the current board.
5. **Import Map:** Allows to load a file with an existing configuration.

In the second column of the page, we have placed the board using a canvas object. It is divided into a matrix where each cell can be clicked and customized except the perimeter that it is not allowed to be modified. If the map would be higher than the space allowed, we have enabled the scroll in both directions: horizontal and vertical.

Finally, in the third column, located at the right of the screen, we can find all the available parameters of the game that can be modified. For example, the size of the board, the regeneration time for each bonus, the maximum scope value or collected stones allowed.

In the figure 5.1 we provide the screenshot with the elements we have explained.
Javascript file: This file includes the code that computes each action done in the Map Editor. Its aim is to provide the correct configuration of the board when the user exports his work. This tool is the first version of the Map Editor, and there are some restrictions as the number of Poquémon per player. This tool is adapted to the game we have created and may need some modifications for the future versions of the game.

There are some important features we have developed for this tool that we want to comment:

**Board’s resize:** When the user changes the width or height of the board, the map clears itself and force us to restart again from the beginning. We reached the way to add or remove rows/columns without losing any data. The cells are added or deleted from the right or the bottom of the board depending if we are modifying the width or the height option.

**Import and Export files:** The export option is the essential feature of the tool. Without it, nothing makes sense because the code of the map configuration would not be available. On the other side, the import file was implemented with the idea of creating a kind of *save/load* options.
Users are now able to export their maps without finish them and load them again whenever they want.

**Check Balancing**: This feature is not visible when the user is creating a new Map. The aim of this control is to restrict the quality of the exported maps. We expect the exported maps are balanced\(^1\) with the bonus for all the players. In the next section, we will deep in this algorithm.

## 5.3 Check Balancing Algorithm

As we introduced, the aim of this algorithm is to avoid the creation of maps with an initial advantage of one or more Poquémon. This checking was developed to reach a powerful tool than initially thought. It will be able to analyse and judge each map depending the number of bonuses it has and where they are placed. The first decision was to implement an algorithm following the *Backtracking* \([18]\) algorithm.

\[
\textbf{int} \quad \text{compute}(i, j, \text{distRemaining});
\]

\[
\begin{align*}
\text{var max} &= (\text{distRemaining}+1) \cdot v + \text{MAX( compute function result in the four adjacent cells )} \\
\end{align*}
\]

Figure 5.2: **Balancing Algorithm: Recursive call**

In the figure 5.2, we provide the header of the algorithm we have implemented and the recursive call. The parameter *distanceRemaining* is the number of rounds left to finish the algorithm and the variable *v* is the bonus value of the cell \(i, j\). Each parameter of *max*’s operation represents the value in the four available directions we can move a Poquémon: *top*, *bottom*, *left* and *right*.

There are a lot of parameters to analyse and evaluate, so we have to take the most important and renounce some others. The parameters discarded implied a hard cost in the development of the algorithm. For example, the location and the behaviour of the opponents when we are analysing one Poquémon was one of these discarded parameters. In other words, when we compute how many points of each bonus a

\(^1\)A map is balanced when each player can reach a similar number of bonus and get a similar score with a determined number of rounds.
Poquémon can reach in $X$ rounds, we remove the opponents from the board and we simulate this Poquémon is playing alone. Obviously, the possible attacks of the game during this rounds or the bonus collected for the opponents were not treated.

The following parameters are those we check before to throw the algorithm in a particular cell:

**The cell content:** If the checked cell is empty $v$ parameter would be 0. Otherwise, it would have a value greater than 0 depending the bonus we are looking for and the weight of this bonus. The distance between the initial position of the Poquémon and the position where the bonus is placed is also important and as near it is, the more value will receive the weighting.

**Legal cell:** We check the cell we are going to move is on the board, and we are not moving out of the board or a Wall cell.

**Avoid repeated cells:** A boolean matrix with the board’s size was created to avoid cycles when we expand the different paths in the algorithm.

**Control Walls:** Given a position and the $distanceRemaining$ value we can compute if in the Position $(i, j)$ will contain a wall or not in a concrete round.

After we verify these conditions, the call is launched again in a new position. When the four recursive calls end we have to restore the visited boolean matrix and the value of each cell because another Poquémon will be also computed and we have to use the same values for all of them.

When run this algorithm for each kind of bonus present in the board. We take the maximum and the minimum value for each one and, if the maximum value is greater more than 1.5 time versus the minimum value, an alert will appear notifying which kind of bonus is unbalanced. To export the map, the board must be balanced.
Chapter 6

Conclusions

After finishing the development of this project, we can say we are very satisfied with the work done and we can assert that the objectives we set when this project was started have been reached.

We set as the first objective to create a new environment for the programming assignment of Data Structures and Algorithms and we have done it. The new game is called Poquémon and it follows the same structure as other current games: the Runner on one hand and the Viewer on the other. Both parts include new features and properties that makes this game one unique among existing. The second objective we set and we reached was to extend the body of the assignment developing a new tool, currently unavailable: a Map Editor for the users. The creation of this tool along with the implementation of the ghost walls in the Runner was among the hardest points of the project.

Fortunately, we have finished the whole project, and it is ready to be released the next semester. We wish no bug appears even though the game has been deeply tested. Moreover, We have committed to maintain the project, and we will solve each new problem that could appear. Finally, we hope enjoy the game as the students will do and know the best strategic user in a few months.

Developing this project has been a great experience, and we hope our work could be used as a support material in the future games. We also expect the students to enjoy developing their assignment and appreciate the effort and desire we have invested with this project.
Chapter 7

Future Work

This project is finished, but there are many ways to further improve it. We want to expose some ideas to increase the game and extend it in the future:

• The first idea we had when we developed this project was to create new versions of this game. Let us to provide a couple of them to understand what does mean to develop a new version of the game.

1. **Without the scope bonus:** This version of the game would just with *attack* and *defense* bonuses. The battles would take place when two Poquémon are be located one beside the other.

2. **Without attack and defense bonuses:** In this version of the game we would keep the scope attribute, but the attack of each Poquémon would be 1 and the defense 0. The upgrade of these attribute would not be allowed.

These versions would modify some rules of the original game, but would keep the same essence that the original version has. Each version of the game can be played with one or more than one Poquémon per player.

• The next idea will be associated with the viewer part. This section of the project is almost completed there is always a feature that can be implemented to improve the tool. In this case we have thought about a *speed controller,* that would allow to control the speed of the game we are watching. This feature would let watching the game faster and slower than we can currently do.
This last point will be dedicated to the improvements of the Map Editor. It is the newest tool of the project, and there are some features we discarded to deliver the project in the given time:

1. The **drag and drop** is the first feature this tool needs in the future. This action allows the user to draw the same object into multiple cells with one single action instead of $N$ actions we need now. It would also be useful to remove multiple objects with just one action.

2. **Select an area**, would be the second feature to add to the editor. This feature is though to move a set of cells from one point to another without losing the structure and the content it has.

3. The last feature we want to consider for a future update is a **game simulation** option. This option would give you the chance to play a game with the developed map. The user would load the desired players into the map editor, and then would click the option *play game* and finally, a new tab with the game will show the result.

   This simulation is just a draft idea because we had not the chance to integrate the Viewer into the map editor.

All these features have been thought to make easier the development of new mazes and help to save as much time as possible. The last point also has the intention to help the testing of the player and improve in a faster way development of the assignment.
Appendix A

Dependencies

On this appendix we will post the dependencies of each phase of the project. Each figure of this appendix represents a single stage of the game and they are the same as we exposed in the planning section.

![Diagram](image)

Figure A.1: *Before the game dependencies.*
Figure A.2: Runner dependencies.
An environment for a programming practical work based on gamification

Figure A.3: *Viewer dependencies.*

Figure A.4: *Map Editor dependencies.*
Appendix B

Gantt Diagrams

In this appendix we have included the Gantt diagram for each phase of the project. There are 4 Gantt diagrams in total. Each diagram starts where the previous diagram finishes. So, if we put all the diagrams next to each other we will be able to see the whole diagram.

![Gantt Diagram](image)

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Jun 22</th>
<th>Jun 29</th>
<th>Jul 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st meet with supervisors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Think about the topic game</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write the draft with the rules</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Share draft to supervisors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solve mistakes after meeting them</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure B.1: The planning of Before the Game phase.
An environment for a programming practical work based on gamification

Figure B.2: The planning of the Runner phase.
An environment for programming practical work based on gamification

Figure B.3: The planning of the Viewer phase.

Figure B.4: The planning of the Map Editor phase.
Appendix C

Game Documentation

C.1 Rules

Poquémon is a game based on the Japanese video game saga Pokémon[6].

Each player controls a number of Poquémon. The goal of a player is to get the maximum score by collecting point bonuses and killing opponent Poquémon.

A match of the game consists of a number nb_rounds() of rounds. In each of these rounds, Poquémon can move around a rectangular board. Cells in this board may be occupied by a Poquémon or contain bonuses of several kinds, or be a wall (which Poquémon cannot cross), or be empty.

Whenever a Poquémon moves to a cell with a bonus (for instance, a point bonus), it collects it; in this case, after some rounds the bonus appears again in a random cell of the board.

The board will always be surrounded by walls. Some walls may appear and disappear along the game. These are called ghost walls and cannot be placed in the perimeter. These ghost walls are located on the board and may be present or hidden. Every wall_change_time() rounds every ghost wall will change its state (present, or not present) but it is not necessary that all board’s ghost walls change at the same round. In other words, they have the same period but not the same phase. If a
Poquémon is located in a position where a ghost wall appears, it dies. To know if there is a ghost wall at a certain position $P$ you can use $\text{ghostWall(}\text{Pos } p\text{)}$ function, which returns the remaining rounds to change the state of the wall or -1 if this cell is not a ghost wall.

In each round, a Poquémon can also attack another Poquémon and wage a battle. As a result, the attacked Poquémon can die. Each Poquémon has some attributes that are considered in a battle: attack, defense and scope. These attributes can be improved by collecting respective bonuses (attack bonus, defense bonus, scope bonus and stone bonus, a special bonus that improves all Poquémon’s stats). The scope of a Poquémon and the number of stones can be collected are limited.

Any dead Poquémon will appear again after $\text{player\_regen\_time()}$ rounds in a position that guarantees that an action can be performed in the next round.

As pointed out above, there are two ways to get points. A player can collect point bonuses of the board or kill opponent Poquémon and win the $\text{battle\_reward()}$ percent of the total points of the opponent.

• The following list explains the parameters that configure a game:

  - $\text{nb\_players()}$: Number of players.
  - $\text{nb\_pokemon()}$: Number of Poquémon per player.
  - $\text{nb\_rounds()}$: Number of rounds in the game.
  - $\text{nb\_ghost\_wall()}$: Number of ghost walls on the board.
  - $\text{nb\_point()}$: Number of point bonuses on the board.
  - $\text{nb\_stone()}$: Number of stone bonuses on the board.
  - $\text{nb\_scope()}$: Number of scope bonuses on the board.
  - $\text{nb\_attack()}$: Number of attack bonuses on the board.
  - $\text{nb\_defense()}$: Number of defense bonuses on the board.
  - $\text{player\_regen\_time()}$: Time (in rounds) before a Poquémon appears again after dying.
  - $\text{wall\_change\_time()}$: Time (in rounds) before a ghost wall changes its status, present or hidden.
point_regen_time(): Time (in rounds) before a point bonus appears again after having been taken.

stone_regen_time(): Time (in rounds) before a stone bonus appears again after having been taken.

scope_regen_time(): Time (in rounds) before a scope bonus appears again after having been taken.

attack_regen_time(): Time (in rounds) before an attack bonus appears again after having been taken.

defense_regen_time(): Time (in rounds) before a defense bonus appears again after having been taken.

battle_reward(): Percent of points the attacker will get of the total points of the defender if it wins the battle.

max_scope(): Maximum scope that a Poquémon can reach.

max_stone(): Maximum number of stones bonuses that a Poquémon can take.

rows(): Number of rows of the board.

cols(): Number of columns of the board.

• The different kinds of cells of the board are:

  Empty: Empty cell.

  Wall: Cell with a wall (ghost wall or not).

  Point: Cell with a point bonus.

  Stone: Cell with a stone bonus.

  Scope: Cell with a scope bonus.

  Attack: Cell with an attack bonus.

  Defense: Cell with a defense bonus.

• Each cell can be visited by at most one Poquémon. Each Poquémon can be alive or (temporarily) dead.

• The first round is the round 0.
• Initially all *Poquémon* will have one point of attack, one point of defense and one point of scope. These attributes can be upgraded by collecting their respective bonuses and will be used to win battles against opponent *Poquémon*.

• Each round, each player can ask only for one action for each of their *Poquémon*. This player can choose—Independently of the other players—what their *Poquémon* have to do: moving to an adjacent position, throwing an attack *OR* nothing. An attack will only be accepted if when it is thrown, there is an opponent *Poquémon* to receive it. Otherwise, the action will be considered as null. If a player asked for more than one action with one of their *Poquémon*, only the first one will be accepted.

• The available directions to move and attack are *top*, *bottom*, *left* and *right*. A *Poquémon* cannot move to a cell with a wall.

• If a *Poquémon* tries to go to a cell occupied by another *Poquémon*, the movement will not be performed.

• In *Poquémon*’s game there are the following kinds of bonuses:

  – Point: Increases player’s score.
  – Attack: Increases *Poquémon*’s attack attribute.
  – Defense: Increases *Poquémon*’s defense attribute.
  – Scope: Increases *Poquémon*’s scope attribute.
  – Stone: Increases some *Poquémon*’s attributes.

• To take a bonus from the board, it is only necessary to move a *Poquémon* to this cell of the board.

• Which are the consequences of collecting each bonus?

  – If a *Point* bonus is collected, the player adds to their scoreboard the value of this bonus. The value of the point bonus can be asked by using `pointsValue(Pos p)` function, which returns the number of points of this cell (100, 200, 300, 400 or 500) or -1 if there are not any point bonuses.
  – If an *Attack* or *Defense* bonus is collected, the *Poquémon* will receive one point of the corresponding attribute.
If a Scope bonus is collected, the Poquémon will receive one point of scope except if its scope is $== \text{max\_scope()}$. In this case, this bonus will not have any effect. Scope lets a Poquémon attack farther. The value of this attribute is the number of cells away a Poquémon can attack.

If a Stone bonus is collected, the Poquémon will receive two points of attack, two points of defense and one point of scope (the latter only if the scope of the poquémon is $< \text{max\_scope()}$). The maximum number of Stones a Poquémon can collect is $\text{max\_stone()}$. When a Poquémon collects more Stone than $\text{max\_stone()}$ this Poquémon will not increase any attribute.

- Each bonus will appear again on the board in a random cell after $\text{point\_regen\_time()}$, $\text{stone\_regen\_time()}$, $\text{scope\_regen\_time()}$, $\text{attack\_regen\_time()}$ or $\text{defense\_regen\_time()}$ rounds, respectively.

- A Poquémon can only see opponent Poquémon when they are at the same row or column and there are not any walls between them.

- When two Poquémon are aligned at the same row or column, and there is not any wall between them, a battle can take place if one of them asks for it. If the distance between the attacker and the defender is larger than the scope of the attacker, the battle does not take place.

If there are more than two Poquémon aligned and the scope of the attacker is enough to hit more than one opponent, only the closest one will receive the attack.

The result of a battle will be computed following the next rule:

Let $a$ be the attack attribute of the attacker, and $d$ the defense attribute of the defender.

If $a \geq d$ then the attacker’s attack updates to $\max(1, \text{attack}-1)$ and wins $\text{battle\_reward()}$ percent points of the total points of the defender (rounding down). The defender dies keeping the same attributes ($\text{attack}$, $\text{defense}$, $\text{scope}$)
An environment for a programming practical work based on gamification

and the same score.

On the other hand, if $a < b$, the attacker’s attack updates to $\max(1, \text{attack}-1)$, the defender’s defense updates to $\max(1, \text{defense}-1)$ and the game continues.

- Only one attack can be executed for each round. In the situation that more than one \textit{Poquémon} asked to attack, the final attacker will be decided randomly.

- The actions requested by the players will be executed in the following way: Firstly we will determine a random order of execution among all players. Then, following this order, the actions will be executed. If a \textit{Poquémon} attacks another \textit{Poquémon} that has moved before and the scope of the attacker is not large enough the attack will fail and the attacker will not do any action.

- At the end of each round the score of each player will be updated and bonuses and \textit{Poquémon} will be regenerated if appropriate. Regenerated bonuses will appear in a random position and regenerated \textit{Poquémon} in a random safe position (a position where the regenerated \textit{Poquémon} cannot find opponent \textit{Poquémon} for at least one round). Finally, the \textit{Ghost walls} with attribute time $== 0$ will change their state.

- When the game is over, the player with the highest score will be the winner.
C.2 Viewer

In the following image we can see a screenshot with most of the elements that are present in the game:

- On the top of the window, we can see some buttons that will allow us to pause/play, go to the beginning and go to the end of the game, deactivate the animation mode or close the viewer. A horizontal slide indicates the round number the game is. A help window will be opened by clicking 'h'. This help explains the keyboard shortcuts that control the viewer.

- The scoreboard is on the left and the right of the board. Each player has his name and avatar. The scoreboard indicates his score, the consumed CPU and his attribute status: Attack, Defense, and Scope (when the player becomes froze, a red 'OUT' appears).

- A fine circle surrounding a Poquémon means that the Poquémon is resurrecting.

- In the screenshot, the red Poquémon is attacking.

- Other elements that appear in the screenshot:
  - Attack
  - Defense
  - Scope
  - Stone
  - Points (with the printed value)
C.3 Map Editor

In the following image we can see a screenshot of the map editor:

- On the right of the window, we can modify the size of the board. Change the regeneration penalty of the *Poquémon* and the bonuses of the game. And customize the game settings as the total rounds, the period of the ghost walls, the percent battle reward or the maximum scope and collected stones allowed.

- To fill the map, we must select a *Poquémon*, bonus or wall from the tiles matrix and placed on the board. Drag and drop are not allowed so you must click each cell INDIVIDUALLY.

- You can only place one *Poquémon* of each.

- You can replace the content of a cell by place another tile in the same position.

- The *Selected Tile* menu shows the tile you will place at this moment.

- To edit the initial settings of a ghost wall you must click the *Tile Selector* button and then click the ghost wall. The menu we can see in the following image will appear. You must CLICK AGAIN the *Tile Selector* button to place more tiles on the board. Otherwise you would not be able.
• To remove a tile and clear the cell you must use click the *Tile Eraser* button. You can remove whatever you want except the perimeter.

• To start again the map you can use the *Clear Board* button. You will keep the board size and the right column settings. To begin the whole map you can refresh the page.

• To export the map you must click *Export Map* button. If your map is unbalanced you will be advised to solve it. To save the exported map, you must copy the content of the newly generated tab into a new file and name it as example.res

• To import an existing map you can click the *Choose a file* button you can find at the last position.
C.4 Programming

The first thing you should do is to download the source code. This source code includes a C++ program that runs the matches and also an HTML5/Javascript viewer to watch them in a nice animated format. Also, a "Demo" player is provided to make it easier to start coding your own player.

C.4.1 Running your first match

Here we will explain how to run the game under Linux, but a similar procedure should work as well under Windows, Mac, FreeBSD, OpenSolaris... The only requirements on your system are g++, make and a modern browser like Mozilla Firefox or Chromium.

To run your first match, follow the next steps:

1. Open a console and cd to the directory where you extracted the source code.

2. Run make all to build the game and all the players. Note that the Makefile will identify as a player any file matching the expression "AI*.cc".

3. The call to make should create an executable file called Game. This executable allows you to run a match as follows:

   ./Game Demo Demo Demo Demo < default.cnf > default.res

   Here, we are starting a match with 4 instances of the player "Demo" (included with the source code), with the game configuration defined in "default.cnf". The output of this match will be stored in "default.res".

4. To watch the match, open the viewer (viewer.html) with your browser and load the "default.res" file.

   A script run.sh for carrying out steps 2-4 automatically is also provided.

   Use the --help option of Game to see a list of all options you can use. For instance, the option --list will show a list with all the available player names.

   If needed, remember you can run make clean to delete the executable and all object files and start over the build.
C.4.2 Adding your player

To create a player, copy the file AINull.cc (an empty player that is provided as a template) to a new file with the same name format (AIWhatever.cc).

Then, edit the file you just created and change the playername line to your own player name, as follows:

```
#define PLAYER_NAME Whatever
```

The name you choose for your player must be unique, non-offensive and less than 12 letters long. It will be used to define a new class PLAYER_NAME, which will be referred to below as your player class. The name will be shown as well when viewing the matches and on the website.

Now you can start implementing the method `play()`. This method will be called every round and is where your player should decide what to do, and do it. Of course, you can define auxiliary methods and variables inside your player class, but the entry point of your code will always be this `play()` method.

From your player class you can also call functions to access the board state, as defined in the `Board` class in `Board.hh`, and to command your units, as defined in the `Action` class in `Action.hh`. These functions are made available to your code using multiple inheritance via the class `Player` in `Player.hh`. The documentation on the available functions can be found in the aforementioned header files of each class. You can also examine the code of the “Demo” player in `AIDemo.cc` as an example of how to use these functions. Finally, it may be worth as well to have a look at the file `Utils.hh` for useful data structures.

Note that you should not modify the `factory()` method from your player class, nor the last line that adds your player to the list of available players.

C.4.3 Playing against the Dummy player

To test your strategy against the Dummy player, we provide the `AIDummy.o` object file. This way you still will not have the source code of our Dummy, but you will be able to add it as a player and compete against it locally.

To add the Dummy player to the list of registered players, you will have to edit the `Makefile` file and set the variable `DUMMY_OBJ` to the appropriate value. Remember
that object files contain binary instructions targeting a specific machine, so we cannot
provide a single, generic file. If you miss an object file for your architecture, contact
us and we will try to supply it.

Pro tip: You can ask your friends for the object files of their players and add them
to the Makefile too!

C.4.4 Restrictions when submitting your player

Once you think your player is strong enough to enter the competition, you should
submit it to the Jutge.org website (https://www.jutge.org). Since it will run in a
secure environment to prevent cheating, some restrictions apply to your code:

• All your source code must be in a single file (AIWhatever.cc).

• Your code cannot use global variables (use attributes in your class instead).

• You are only allowed to use standard libraries like vector, map, cmath...

• Your code cannot open files nor do any other system calls (threads, forks...).

• Your CPU time and memory usage will be limited when executed on Jutge.org.
The time limit is 1 second for the execution of the entire game. If the time limit
has been exceeded (or if the execution of your code aborts), your player will be
frozen and will not admit further instructions any more.

• Your program should not write to cout nor read from cin. You can write debug
information to cerr (but remember that doing so on the code you upload can
waste part of your limited CPU time).
C.5 Tips

- Read only the headers of the classes in the provided source code. Do not worry about the private parts nor the implementation.

- Start with simple strategies, easy to code and debug, since this is exactly what you will need at the beginning.

- Define basic auxiliary methods, and make sure they work properly.

- Try to keep your code clean. Then it will be easier to change it and to add new strategies.

- As usual, compile and test your code often. It is much easier to trace a bug when you only have changed few lines of code.

- Use `cerrs` to output debug information and add `asserts` to make sure the code is doing what it should do. Remember to remove (or comment out) the `cerrs` before uploading your code to Jutge.org, because they make the execution slower.

- When debugging a player, remove the `cerrs` you may have in the other players’ code, to make sure you only see the messages you want.

- By using commands like `grep` in Linux you can filter the output that `Game` produces.

- Switch on the `DEBUG` option in the Makefile, which will allow you to get useful backtraces when your program crashes. There is also a `PROFILE` option you can use for code optimisation.

- If using `cerr` is not enough to debug your code, learn how to use `valgrind`, `gdb`, `ddd` or any other debugging tool. They are quite useful!

- You can analyse the files that the program `Game` produces as output, which describe how the board evolves after each round.

- Keep a copy of the old versions of your player. When a new version is ready, make it fight against the previous ones to measure the improvement.
• Before competing with your classmates, focus on qualifying and defeating the "Dummy" player.

• Make sure your program is fast enough: the CPU time you are allowed to use is rather short.

• Try to figure out the strategies of your competitors by watching matches. This way you can try to defend against them or even improve them in your own player.

• **DO NOT GIVE YOUR CODE TO ANYBODY.** Not even an old version. We are using plagiarism detectors to compare pairwise all submissions (including programs from previous competitions). However, you can share the compiled .o files.

• Do not wait till the last minute to submit your player. When there are lots of submissions at the same time, it will take longer for the server to run the matches, and it might be too late!

• Most of the game parameters (number of rounds, ...) will not change, but if your strategy can adjust to them, you will be extra-safe in case some changes are needed.

• You can submit new versions of your program at any time.

• If you create your own board for the game, send it to us before the competition starts and maybe we will include it!

• And again: Keep your code simple, build often, test often. Or you will regret.
Bibliography


