



Pedro Igor Porfírio Sampaio

**A Study on Pervasive Games Based on the
Internet of Mobile Things**

Dissertação de Mestrado

Dissertation presented to the Programa de Pós-graduação em Informática da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor : Prof. Bruno Feijó
Co-advisor: Prof. Markus Endler

Rio de Janeiro
October 2018



Pedro Igor Porfírio Sampaio

**A Study on Pervasive Games Based on the
Internet of Mobile Things**

Dissertation presented to the Programa de Pós-graduação em Informática da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática.
Approved by the undersigned Examination Committee.

Prof. Bruno Feijó

Advisor

Departamento de Informática – PUC-Rio

Prof. Markus Endler

Co-advisor

Departamento de Informática – PUC-Rio

Profª. Noemi de La Rocque Rodriguez

Departamento de Informática – PUC-Rio

Augusto Cesar Espíndola Baffa

Centro Técnico Científico – PUC-Rio

Prof. Márcio da Silveira Carvalho

Vice Dean of Graduate Studies

Centro Técnico Científico – PUC-Rio

Rio de Janeiro, October 2nd, 2018

All rights reserved.

Pedro Igor Porfírio Sampaio

The author has graduated in Computer Science by the University of PUC-Rio (Rio de Janeiro, Brasil) in the year 2015, enrolling for a Master's Degree in Computing at PUC-Rio in the year 2016.

Bibliographic data

Sampaio, Pedro Igor Porfírio

A study on pervasive games based on the internet of mobile things / Pedro Igor Porfírio Sampaio ; advisor: Bruno Feijó ; co-advisor: Markus Endler. – 2018.

119 f. : il. color. ; 30 cm

Dissertação (mestrado)–Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2018.

Inclui bibliografia

1. Informática – Teses.
 2. Jogos pervasivos.
 3. Internet das coisas.
 4. Internet das coisas móveis.
 5. Jogos mobile.
 6. Sensoriamento participativo.
- I. Feijó, Bruno. II. Endler, Markus. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. IV. Título.

CDD: 004

To my parents, family and friends, for their support
and encouragement.

Acknowledgments

I would like first to thank my advisor Bruno Feijó and co-advisor Markus Endler for the support and for guiding me through all the work. In addition, I want to thank all professors that were part of my formation, in special Ivan Filho, Luiz Seibel, Noemi Rodriguez, Simone Barbosa and Edward Hermann.

I would also like to thank my parents Pedro Porfírio and Fátima Curvelo, for all the encouragement and love. Sadly, my father was not able to see the final results of this work, but I will never forget all the support I have throughout my whole life.

I want to thank my girlfriend Lívia Moreno, my brother Pedro Ivo and his girlfriend Lívia Mezavila, my grandmother Waldiceia Curvelo, my uncles Arenildo dos Santos and Pedro Sérgio, my aunts Patrícia Curvelo and Monica Reis and my cousins Isabelle Curvelo, Caroline Curvelo, Beatriz Reis and Letícia Reis for all the love and support.

Additionally, I want to thank all my friends, in special Matheus Arnaut, Marcos Arnaut, Bruno Cister, Ruben Perorazio, Raphael Vassal, Rafael Cipriani, Felipe Cipriani, Thiago Cipriani, Walter Trajano, Angelo Gomes e Breno Costa, for the support and friendship throughout my whole life. A special thanks to Luiz Pitta, Fernando Abreu, Lucas and Augusto Baffa for always providing me with advice that helped me during my time in the University.

Finally, I want to thank my colleagues of the laboratories ICAD and LAC. In special, I want to thank José Paulo, Luís Fernando, Douglas Silva and Felipe Carvalho for helping me throughout my research. Also, I want to thank Leonardo Abreu, Rafael Damázio, Daniel Zimmer, Jhonatha Neves and Ana Clara.

Lastly, I would like to thank all the people from the Department of Computing, in special Vagner Pires, Marcelo Alves, Marcos Pawlowski, Regina Zanon and Cosme Pereira.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Abstract

Porfírio Sampaio, Pedro Igor; Feijó, Bruno (Advisor); Endler, Markus (Co-Advisor). **A Study on Pervasive Games Based on the Internet of Mobile Things.** Rio de Janeiro, 2018. 119p. Dissertação de mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Mobile pervasive games are a game genre that combines the real and virtual worlds in a hybrid space, allowing interactions with not only the virtually created game world, but also with the physical environment that surrounds the players. The Internet of Mobile Things (IoMT) specifies situations in which devices on the Internet of Things (IoT) can be moved or move autonomously, while maintaining remote connectivity and accessibility from anywhere on the internet. Following the huge success of recent mobile pervasive games and the coming IoT boom, we provide an integration for all the technology involved in the development of a mobile pervasive game that incorporates IoT devices. We also propose a mobile pervasive game that evaluates the benefits of the union of both fields. This game prototype explores ways of increasing the experience of players through pervasive mechanics while taking advantage of the player's motivation to perform sensing tasks. It also incorporates serious applications into the gameplay, such as the localization of facilities and services.

Keywords

Pervasive Games; Internet of Things; Internet of Mobile Things; Mobile Games; Participatory Sensing.

Resumo

Porfírio Sampaio, Pedro Igor; Feijó, Bruno; Endler, Markus. **Um Estudo Sobre Jogos Pervasivos Baseados na Internet das Coisas Móveis.** Rio de Janeiro, 2018. 119p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Jogos pervasivos móveis são jogos que combinam os mundos real e virtual em um espaço híbrido, permitindo interações não apenas com o mundo do jogo virtualmente criado, mas também com o ambiente físico que envolve os jogadores. A Internet de Coisas Móveis (IoMT) especifica situações em que os dispositivos na Internet das Coisas (IoT) podem ser movidos ou se moverem de forma autônoma, mantendo conectividade remota e acessibilidade de qualquer lugar na Internet. Segundo o enorme sucesso dos recentes jogos pervasivos móveis e a iminente expansão de IoT, nós fornecemos uma integração para toda a tecnologia envolvida no desenvolvimento de um jogo pervasivo móvel que incorpora dispositivos IoT. Também propomos um jogo móvel pervasivo que avalia os benefícios da união de ambos os campos. Este protótipo de jogo explora maneiras de aumentar a experiência dos jogadores através de mecânicas pervasivas, aproveitando a motivação dos jogadores para realizar tarefas de sensoriamento. O jogo também incorpora aplicações sérias na jogabilidade, tais como a localização de instalações e serviços.

Palavras-chave

Jogos Pervasivos; Internet das Coisas; Internet das Coisas Móveis;
Jogos Mobile; Sensoriamento Participativo.

Table of contents

1	Introduction	15
1.1	Main Challenges	16
1.2	Motivation and Goals	17
1.3	Contributions	18
1.4	Methodology	19
1.5	Document Structure	19
2	Related Work	20
3	Overview	23
3.1	The Game	23
3.2	Pervasive Mechanics	24
4	Development and Runtime Environment	29
4.1	Android Studio	29
4.2	ContextNet Middleware	29
4.3	LibGDX Framework	30
4.4	2DMapBuilder Map Editor	31
4.5	Gradle Build Tool	32
5	PUCmon Development	33
5.1	Client-Server Architecture	33
5.1.1	Communication Architecture	33
5.1.1.1	Message Classification	33
5.1.1.2	Message Distribution	34
5.1.2	Database Architecture	36
5.1.2.1	Database Model	36
5.1.2.2	Database Integration	37
5.1.3	Account Management	38
5.2	LibGDX Coordinate System	39
5.2.1	Touch Coordinates	40
5.2.2	Screen Coordinates	40
5.3	Screen State Architecture	40
5.3.1	LibGDX Game Loop	41
5.3.2	State Design Pattern	41
5.4	Multiple Language Support	43
5.5	Configuration File	45
5.6	Text System	45
5.7	Sound Effects	46
5.8	Map	47
5.8.1	2DMapBuilder Integration	48
5.8.2	Optimized Map Rendering	49
5.9	Game's Camera	51
5.10	Animation Architecture	52

5.11 Collision Detection	54
5.11.1 Collision Architecture	55
5.11.2 Collision Mapping	57
5.11.3 Optimized Collision Check	58
5.12 Character	58
5.12.1 Character's Level System	58
5.12.2 Character's Attributes	60
5.12.3 Character's Inventory	60
5.13 Items	61
5.14 Enemies	63
5.15 Battle	64
5.16 Ranking	66
5.17 Market	67
5.18 Pervasive Map	68
5.19 Sensor Search	70
 6 Evaluation	 72
6.1 Experiments with Students	72
6.1.1 Experiment A	72
6.1.2 Experiment B	75
6.2 Performance Tests	78
6.2.1 Latency Tests	80
6.2.2 Rendering Performance	81
 7 Conclusions	 83
7.1 Future Works	85
7.2 Recommendations	87
 Bibliography	 89
 A Setup Guide	 94
A.1 Android Studio Setup	94
A.2 LibGDX Project	94
A.3 ContextNet Setup	95
A.3.1 SDDL Core Setup	96
A.3.2 Mobile Hub Setup	97
A.4 Gradle Setup	99
A.5 Database Setup	100
 B Android Manifest File	 104
 C Gradle Build Files	 107
C.1 Root Folder Gradle File	107
C.2 Android Folder Gradle File	109
C.3 Core Folder Gradle File	113
 D SQL Schema File	 115
 E Data Source Factory	 119

List of figures

Figure 3.1	Experimental game created for the research	23
Figure 3.2	Experimental game missions system screens	25
Figure 4.1	Experimental game client-server and sensor communications model	30
Figure 4.2	Example map in 2DMapBuilder map editor	31
Figure 5.1	Messages classification architecture illustration	34
Figure 5.2	Message distribution architecture illustration	35
Figure 5.3	Database Model generated in phpMyAdmin	37
Figure 5.4	Database properties file example	38
Figure 5.5	Account creation screens	38
Figure 5.6	Y-down Coordinate System	39
Figure 5.7	State architecture overview	43
Figure 5.8	English and Portuguese property files	44
Figure 5.9	Game's BGM in FamiTracker software	46
Figure 5.10	2D game camera illustration	51
Figure 5.11	2D sprites example	53
Figure 5.12	Animated entity state machine	54
Figure 5.13	Collision system illustration	56
Figure 5.14	2DMB collision tool usage	57
Figure 5.15	Level progress chart illustration	59
Figure 5.16	Character attributes example	60
Figure 5.17	Fully equipped character inventory	61
Figure 5.18	Golden Skull Legs effects at level 25	63
Figure 5.19	Enemy with random traits generated for battle	64
Figure 5.20	Battle screen showing the fight with a generated enemy	65
Figure 5.21	Steam charts of games with most current players showing Clicker Heroes	66
Figure 5.22	Ranking with the game's top players	67
Figure 5.23	Market buy tab containing showing items being sold	68
Figure 5.24	Native code interfacing scheme	71
Figure 6.1	Chart representing number of men and women in the stand throughout the event	75
Figure 6.2	Temperature time series throughout the event in the stand	76
Figure 6.3	Humidity time series throughout the event in the stand	76
Figure 6.4	Average score of the male bathroom used in the experiment	79
Figure 6.5	Average score of the female bathroom used in the experiment	79
Figure A.1	Android Studio SDK Location configuration	95
Figure A.2	LibGDX Project Generator	95
Figure A.3	Command line that deploys gateway locally	96
Figure A.4	Mobile Hub application	98

Figure A.5	Android manifest file services declaration	98
Figure A.6	Android Studio studio.vmoptions file	99
Figure A.7	Gradle local and remote dependencies	100
Figure A.8	Android libs folder (left) and core libs folder (right)	101
Figure A.9	XAMPP web server with Apache and MySQL services on	101
Figure A.10	The phpMyAdmin page accessed locally	102

List of tables

Table 6.1 Client-server latency tests (ms for milliseconds)	80
Table 6.2 Game screens FPS rendering tests	81

List of Abbreviations

IoT – *Internet of Things*

IoMT – *Internet of Mobile Things*

M-Hub – *Mobile Hub*

M-OBJ – *Mobile Object*

SDDL – *Scalable Data Distribution Layer*

BLE – *Bluetooth Low Energy*

WPAN – *Wireless Personal Area Network*

WWAN – *Wireless Wide Area Network*

GPS – *Global Positioning System*

RPG – *Role-Playing Game*

MMORPG – *Massively Multiplayer Online Role-Playing Game*

2DMB – *2DMapBuilder*

IDE – *Integrated Development Environment*

SDK – *Software Development Kit*

JDK – *Java Development Kit*

DDS – *Data Distribution Service*

MR-UDP – *Mobile Reliable User Datagram Protocol*

IP – *Internet Protocol*

SQL – *Structured Query Language*

LAN – *Local Area Network*

WAN – *Wide Area Network*

URL – *Uniform Resource Locator*

JDBC – *Java Database Connectivity*

API – *Application Programming Interface*

UI – *User Interface*

NPC – *Non-Player Character*

BGM – *Background Music*

MP3 – *Moving Picture Experts Group Layer-3 Audio Format*

OGG – *Ogg Audio Format*

WAV – *Waveform Audio Format*

XML – *Extensible Markup Language*

DOM – *Document Object Model*

FPS – *Frames Per Second*

You should enjoy the little detours to the fullest. Because that's where you'll find the things more important than what you want.

Yoshihiro Togashi, *Hunter x Hunter*, Vol. 32.

1

Introduction

The Internet of Things (IoT) is a network of objects of our day-to-day life that are capable of communicating with each other, aiding in data collection as well as helping people perform tasks, learn, or simply amuse themselves. In this work, we are interested in a new concept proposed and developed by the authors of [1]: the Internet of Mobile Things (IoMT). IoMT refers to scalable situations in which connected objects can be moved or move independently and yet remain remotely accessible and controllable from anywhere on the internet. In [1], the authors team proposed the concept of Mobile Hub (M-Hub) which is a mobile service of middleware responsible for discovering and opportunely connecting to objects that may be stationary or mobile that are accessible only through short-range WPAN technologies. It uses BLE technology for its low energy consumption and for having a reasonable range that floats around fifty meters to one hundred meters. The M-Hub (which runs on smartphones) also provides context information of objects that are in their neighborhood (such as approximate location, if they are in movement or not, what is the luminosity of the environment, ...).

Mobile pervasive games are very different from traditional digital games because they merge the real and the virtual worlds in a hybrid space. In these games, players use mobile devices to move around and interact with elements of the real world, interacting with physical locations and objects, and incorporating real-world properties into the narrative about the game. In this context, the technology that provides the link between the real and the virtual world usually is the GPS (Global Position System), but some works regarding the use of IoT devices such as sensors and actuators have shown the potential of expanding the variety of technologies in pervasive games. Besides enhancing the value of a game, this variety of technologies also increases the possibilities regarding other applications that can be integrated in the pervasive game, such as participatory sensing tasks and other serious applications.

Participatory sensing is the process by which individuals and communities collect and analyze information in a ubiquitous way, using smartphones and cloud services, to form a body of knowledge capable of bringing them to discoveries and to a better quality of life and citizenship [5, 6] e.g. identifying

full trash cans that need to be emptied. Individuals can help gather sensor data in the environment that would be problematic to get automatically through dedicated devices, by simply sharing their internet bandwidth to act as a mean for retrieving data that is collected by a variety of sensors that are in locations that are not connected (or connectable) to the Internet, or simply do not have the required Wireless WAN (3G/4G) connectivity for such operation.

Serious applications refer to the sphere of applications where the main goal is not entertainment of the user. Instead, serious applications aim at providing solutions for real activities that the user go through, striving for automatizing and increasing the efficiency of the realization of such activities. They can range from bank transaction applications to schedule organizer applications. In this project, we focus on serious applications that are related to the activities of students, such as the use of the campus library services.

In the context of pervasive games, where there is an intrinsic motivation that the entertainment aspect brings, we intend to explore the integration of both participatory sensing tasks and serious applications in the game mechanics, analyzing the outcome of the integration in relation to the game experience. We use our University Campus as an area of experiment, where students have several services at disposal, such as libraries and laboratories, and can also contribute by providing reports that better the campus quality of life e.g. defective lamp posts, full trash cans, bathroom malfunctions, broken air conditioners.

1.1

Main Challenges

To integrate all development aspects of a project can be a highly demanding task. When it comes to developing games, the usual first development decision is to choose a game development framework that will help the development process, which can be already a tricky task. Another problem that may arise is the integration of the framework with other libraries used in a project. Pervasive games are a type of game that includes other technologies due to its nature of integrating the real environment with the virtual world. On top of that, pervasive games may include online features that involves client-server communication technologies. Lastly, if we extend the pervasive devices in use, with devices such as sensors and actuators, we also need the technology capable of communication with such devices. The integration of all these components can be very time consuming, leading to extra development costs, as the amount of diversity in the technology hardens the task of integration with no guarantee of the desired result. Thus, to have all the necessary technologies

for this project features, we propose the use of the frameworks and libraries used to achieve our goals, later detailed in this document.

One of the challenges of participatory sensing is engaging people to use the 3G / LTE / 4G bandwidth of their devices and to collect data from distributed sensors in a region in order to contribute to the data collection periodically and with the greatest possible coverage of the region of interest. To aid in achieving this, the user can be rewarded with real-world or virtual incentives, such as points, coins, strength or special abilities of his/her game characters. Many of such incentives are present in games, and if it's a pervasive game, the user (player) may be encouraged to move to specific physical places (such as in games like Pokémon Go) and, thus, contribute to participatory sensing. In short, with the game, the participatory data collection process through sensor visits can consider the inherent motivation in the processes of entertainment, learning and socialization. Regarding this, we speculate that, for example, geographic areas may be well covered by a smaller number of participants if these participants are more mobile, which is in turn achieved by motivating them with game-specific incentives.

1.2 Motivation and Goals

The IoT market has been rapidly growing in the last few years, with huge investments in the area by both small and big companies. This rapid growth shows the potential for a future where everything in our surroundings may become part of the a huge mesh (or network) of IoT smart objects, where some may be mobile, such as tags attached to dogs, mobile robots, keychains, etc. The recent boom of pervasive games, with games with huge success such as Pokémon Go [19], shows that this style of game has great promise that can be explored, resulting in new pervasive games that are constantly being announced. In this project, we use the middleware proposed in [2] to support IoMT applications: ContextNet IoMT, in which we use smartphones as a hub resulting in a cheap connection with smart things that do not have WWAN connectivity. More specifically, we intend to explore new paradigms, methods and techniques of pervasive games combined with the IoMT concept.

In the proposal of this project, we advocate that the combination of pervasive games with IoMT allows for the emergence of a new dimension of participatory sensing, which is the dimension related to entertainment, learning, and socialization. Through the incentives that are present in games, we intend to explore ways of motivating players to perform participatory sensing tasks.

In addition to the main objectives of the game related to entertainment and the knowledge of the university environment, we also explore some serious applications related to the activities of students. In this way, we have the expectation that techniques of presence recording integrated in the game, where the connection with sensors is used to register attendance in classroom, may shine a light on automatic classroom attendance checking.

In summary, this project aims to explore and discuss the possibilities of integration of sensors and actuators of IoMT in a context of pervasive games with the use of mobile devices, exploring the integration of serious applications and participatory sensing in the game's mechanics. For this, we present a RPG (Role-Playing Game) online game that uses ContextNet's IoMT [2] tools to enable communication with IoMT sensors and actuators. We also discuss the whole process of integrating all the pieces (IDE, build tools, frameworks, etc...) that compose the development environment in order to guide future efforts. Thus, we make our integration available as a platform that can be easily adapted to define other games, with other sensors and other examples of participatory sensing and serious applications, as the integration provided enables the full use of a game development framework allied with the ability of communication with IoMT devices through ContextNet middleware.

1.3 Contributions

The main contributions of this dissertation are:

- The field test and evaluation of an experimental pervasive online game that aligns concepts of IoMT and participatory sensing;
- A guide for the integration of the ContextNet's IoMT tools with libGDX framework and other building blocks that composes the development of a pervasive online game with IoMT devices integration;
- Research with students in the PUC-Rio campus where we evaluate the degree of user satisfaction with the game, the possibility of the integration of serious applications (e.g. spatial guide of the campus) and participatory sensing tasks;
- Recommendations based on the experience of developing pervasive mobile games with the ContextNet middleware aligned with IoT devices, serious applications and participatory sensing tasks, such as security recommendations.

1.4

Methodology

Once the experimental game was ready in a suitable state for tests, we prepared experiments that were composed of a sequence of tasks in the game for players to conclude. Each player that was subject of the test completed the whole pack of tasks in the game while being observed. During the test of the game, the test subjects (students) were encouraged to take their time to finish the tasks without any hurry. After completing the tasks, each player headed to a personal interview that assessed their overall experience with the game, as well as their perception of the integration of serious applications and participatory sensing in in-game mechanics. The interview was composed of a sequence of questions related to the game experience, and the answers were all logged to be further analyzed.

1.5

Document Structure

From this point on, this document is organized in the following chapters:

- **Chapter 2 - Related Work:** This chapter provides other works that are related to pervasive games, including works that associates IoT and participatory sensing with these type of games.
- **Chapter 3 - Overview:** In this chapter, an overview of the experimental game features and gameplay, focusing on the pervasive mechanics, is introduced.
- **Chapter 4 - Development and Runtime Environment:** The Development and Runtime Environment chapter gives a detailed explanation of the elements that composes the project.
- **Chapter 5 - PUCmon Development:** This chapter is responsible for providing game developing algorithms and architectures used in the project, as well as detailing some developing aspects of the components used.
- **Chapter 6 - Evaluation:** In this chapter we display and discuss the results obtained on the evaluation of performance and public experiments.
- **Chapter 7 - Conclusions:** This chapter contains the conclusions about this work, as well as some suggested future works and recommendations.

2

Related Work

There are few works relating pervasive games and IoT. A close reference is the one developed in [27], in which the authors propose the combination of interaction and gamification tactics for the design of new experiences of pervasive IoT games.

The use of game design elements in non-game contexts defines the act of gamification [38], and it is extensively used in several different contexts, usually striving for encouraging user motivation. This tactic has also been proposed in the context of participatory sensing, with researches demonstrating how gamification can be used as an incentive to increase user participation on participatory sensing tasks [36, 37].

Another reference related to the objectives of pervasive games associated with IoT and participatory sensing can be found in [7]. The authors of this paper developed a game on the campus of New Jersey Institute of Technology to analyze what kind of incentive can be given so that sensoring tasks (i.e. collecting large amounts of data extracted from the real physical world) are performed with efficiency. This type of task is known as crowdsensing and crowdsearch [8].

The relation between pervasive computing and IoT is analyzed in the special issue of IEEE magazine of 2016, where the author indicates that both communities are pursuing similar use cases, including smart cities, environmental monitoring, agriculture, home automation, and health and wellness monitoring. [9].

Requirements engineering methods for mobile pervasive games are proposed in [3, 4, 10]. According to the proposed work, the designer of a pervasive game should meet the largest number of quality requirements specified by the proposed method, which proposes a series of questions (called checklist) related to different aspects of a pervasive game, such as spatiality, which refers to aspects related to the physical space usage. Designers can use the set of questions to evaluate pervasive game qualities (and thus, pervasiveness) in existing projects [3].

The human sense of territoriality, i.e., the sense of ownership or occupancy of areas and possessions, has been explored in a large number of games,

even before the era of digital games e.g. Risk. In [11], the authors analyze ways in which the sense of human territoriality can be expressed in games that are based on real-world player locations using devices such as GPS. After the development of a pervasive game called City Conqueror, in which the players can conquer physical location in their surroundings, deploy units to defend their territories and attack other players that are in their physical proximity, a survey with players of the game has been conducted. As a result of the survey, the authors concluded that the players prefer to claim real places instead of virtual territories and prefer to play in the real world with people around them. This promising result demonstrates the potential in uniting the physical world around the player with the virtual world of games.

The first successful pervasive game that explored the physical environment around players through GPS devices as an integral part of gameplay was the game Ingress [12]. Ingress is a mobile game based on the physical location of the players and in augmented reality developed by the company Niantic, released in 2012 [13]. Ingress, through its game mechanics, encourages players to explore and travel more, looking for new places and experiences both in the game and in the real world [14]. Games that promotes this behaviour have the potential to incorporate IoT devices and participatory sensing tasks in a smooth and logical manner. Ingress was a huge success, being installed by more than 11 million people in its first two years of existence and having a number of active users around one million in 2015 [15]. Experts estimate that players have collectively walked 258 million kilometers due to the game by 2015 [16]. This kind of impact invites researchers to reflect on how to use pervasive games for serious applications, such as applications for education, culture and crowdsensing.

Pokémon Go has recently reached the apex in terms of success of a pervasive mobile game [17, 18], obtaining the status of most popular game of all time [19]. One of the reasons for this lies in its mechanics that uses GPS devices to integrate physical locations to its gameplay, in a very similar manner compared to the game Ingress. This similarity is not by chance, since Ingress is treated as a precursor of Pokémon Go [20] and both were developed by the company Niantic. Launched in 2016 [17, 18, 20], Pokémon Go has surpassed US\$1 billion in revenue [22] and continues with a large number of players until today, reaching an amount of 65 million players active monthly in 2017 [21]. Naturally much of the success is also due to the popularity that already existed around the series of games Pokémon [20], but it is clear the influence that the use of IoT devices as vital functionality of the game had in capturing the attention and interest of people.

The game Pokémon Go has also generated benefits that were not fully expected in its design such as improved health of sedentary people and positive outcomes with autistic children. Negative results, such as accidents, also deserve special attention (there are several reports in the media about such accidents). The company Niantic even developed a wearable device (Pokémon Go Plus) that allows the player to do some actions (e.g. capture a Pokémon, with some limitations) without using your smartphone. In addition, a newer version of Pokémon Go now includes a speed limit. However, these improvements are still far from solving the main issues with security and social rightness. This question of unintended consequences in pervasive game design (with special mention to the Pokémon Go) can be found in an article of the laboratory ICAD / VisionLab [10].

New pervasive games are being announced every day, taking advantage of the recent success and attention achieved by games of the genre. The success of Pokémon Go did not stop the ambitions of the Niantic company, which announced a new pervasive game, this time based on the universe of the famous series of books and movies Harry Potter. The game, entitled Harry Potter: Wizards United, is set to launch in 2018, however details about its gameplay are scarce [23]. Another famous series that will also serve as universe for a new pervasive game is the comic book and TV series Walking Dead. The game, called Walking Game Dead: Our World, will allow players to fully immerse themselves into the action of the TV series, but there is still no launch forecast [24]. Finally, Universal announced the game Jurassic World Alive, based on the famous series of movies Jurassic Park, which will have a similar model to Pokémon Go, only this time players will be collecting dinosaurs [35].

Having the pervasive game and IoT literature as background, as well as the recent released pervasive games popularity, we designed an experimental online pervasive mobile game that expands beyond the use of GPS devices as an integral part of the gameplay, by integrating short-range WPAN compatible devices, such as sensors and actuators, in the game as a core feature of the gameplay, exploring the possibilities and challenges of this approach, particularly with participatory sensing and serious applications integration.

3 Overview

This chapter provides an overview of the experimental game's main features and gameplay, highlighting its pervasive mechanics. Giving a brief illustration of the game's universe, we intend to better elucidate how the game combined with its pervasive features works, while showing benefits and the potential for using IoT devices in pervasive games.

3.1 The Game



Figure 3.1: Experimental game created for the research

Figure 3.1 shows the experimental game developed for Android smartphones, a 2D tile-based role-playing game (RPG) game inspired in the popular RPG series Pokémon [28], and the object of our research. To aid in the development of the game scenario, the 2D tile-based map editor 2DMB [29] was used, since it was created with this exact purpose.

As stated before, the game also has online features that strive for increasing players experience and motivation introducing competitive and cooperative

mechanics between all players, in the molds of MMORPGs (Massive Multi-player Online Role-Playing Games), a genre known for its ability to encourage intrinsic motivation and prolonging the interest in the game [30]

With features such as infinite character evolution, infinite variation of character equipment and enemies, battle system based on idle games [31], mission system inspired in popular RPG games, a detailed RPG based attributes system that defines character strengths, allied with the online mechanics, such as visualization of the top characters and item market place where players can buy and sell items found in their adventure, we intended to create a game universe capable of motivating and earning players interest, where the goal relies on a continuous quest for becoming stronger by battling enemies and completing missions, pursuing the top spot in the ranking of game players. In this universe, we introduce the use of IoT devices and the pervasive aspect of the game through the mission system, providing a natural integration that is explained in the next section.

3.2 Pervasive Mechanics

We wanted to introduce the pervasive features in the most natural way accordingly to the game characteristics, with the goal of not getting away from the game's RPG roots, in order to be able to integrate pervasive sensing tasks into the gameplay in a plausible manner.

One of the main online RPG gameplay element is the existence of missions that players undergo to advance in the game storyline, offering rewards to players at the end of each mission. This particular artifice was one of the traits that influenced in our game style choices, as we considered it a very flexible and straightforward way of integrating pervasive elements, including participatory sensing and serious applications tasks.

With that in mind, we developed a pervasive mission system that is illustrated in figure 3.2. A mission is directly related to a real location in the game's pervasive map, that is, in our experiment, the campus of our university. Each mission is associated to an IoT device. The general flow of the mission system consist in:

1. Finding the IoT device associated to the mission through the mission description and the map guide, as exemplified in the first image of figure 3.2;
2. Accessing the found IoT device through the sensor interaction screen of the game, as shown in the second image of figure 3.2;

3. Following in-game instructions (third image of figure 3.2) to get the mission's reward, such as experience points, in-game currency and character equipment.

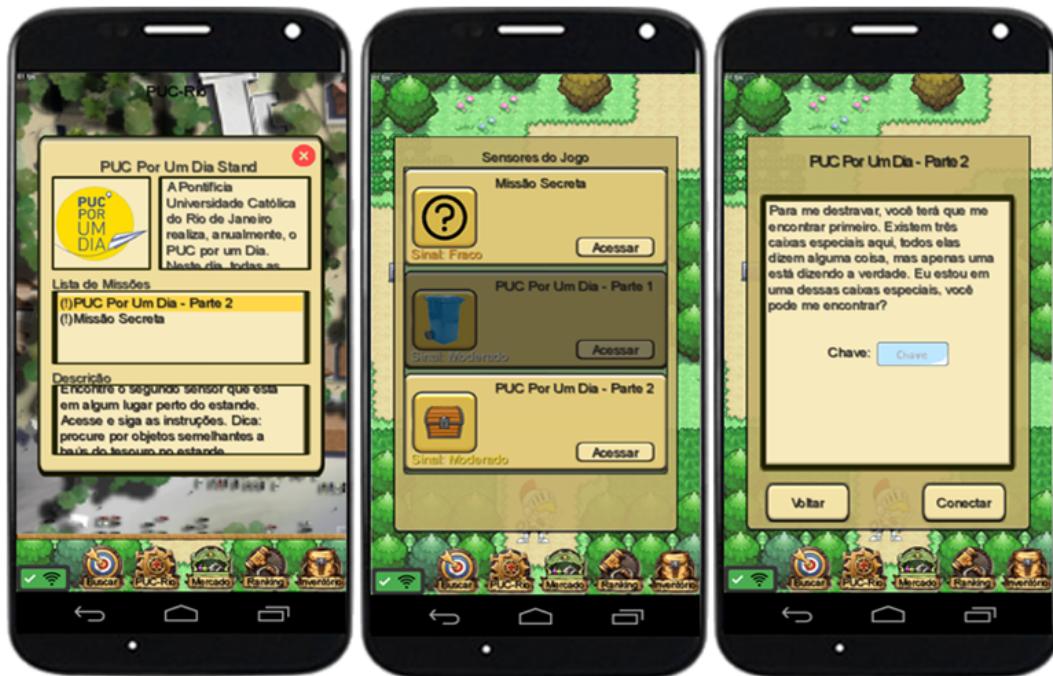


Figure 3.2: Experimental game missions system screens

In the system, a mission can precede other missions and be repeatable, which means that the player can repeat it after a certain amount of time. Also, the mission reward may be locked by a code (as shown in the third image of figure 3.2) and/or request player inputs if desired. Those characteristics gives more flexibility to the integration of pervasive duties. To better illustrate how these behaviours help the integration, some possible pervasive missions implementation are detailed hereinafter.

For student class attendance registration, a possible implementation is a mission that rewards players that attend a particular class, where the professor is the one responsible for carrying the mission IoT device and having it enabled during class time. Naturally, there are more than one lecture, so we need to be able to repeat the mission, thus this mission should be repeatable. Moreover, we can try to cover early departures by adjusting the mission interval of time to match somewhat the class length, so students can complete the mission on arrival and on departure. Another approach is to simply measure the time that each student was in the sensor's range. To be able to properly register a student attendance, some kind of student identification is necessary, so there is a need to tie this mission to an input that is related to the identification. One more thing we can do with the mission system in this case is to add a

code to this mission that can help avoiding rewarding students that are not enrolled in the class, since this code can be only disclosed to students of the class. Of course, in addition to fulfilling the attendance quota, each time a student of the class registers presence through this mission, rewards are given in-game, while the student data is sent to the server. Further automation can be researched to optimize the process, such as the automatic connection with sensors without any need of manual operations and the automatic mapping of each student id for presence registering in the database.

There are two layers of data that players can help collecting: data obtained through autonomous IoT devices and data that are not easily obtained through this kind of devices. Either way, the player helps improving the data collection of his surroundings either by sharing his bandwidth to retrieve local sensing data or by additionally being the agent responsible for data collection. Missions in which the attached sensor also collects sensing data should inform the player that he is being responsible for sending the collected data to the servers, and properly acknowledge the contribution. The mission can require extra input from the player if the data is not obtained by the sensor alone, as the player is asked to contribute even more with the data collection. In this case, a suitable in-game reward should be given by the end of the mission. Bear in mind that missions of this kind don't necessarily need to be disassociated from the virtual storyline, as it can be integrated with in-game quests i.e. "to find a suitable place for the plantation, find a place with convenient climate".

Missions related to the spatial guide of the campus are simple to implement in the system. Since the system consists of an pervasive map (as of now, the campus of PUC-Rio) and each mission is tied to a location of the pervasive map, we can add numerous campus locations that we want to be part of the spatial guide of the pervasive map, with each location having their own missions. With that in mind, we are able to create an interactive guide map that encourages the exploration of the campus by rewarding the players on visiting missions completion. For each location, a visiting mission can be added to encourage players to visit the location, helping new students to get to known important campus facilities. This type of mission is conveniently easy to implement with simple proximity sensors, that are usually cheap, and we have a more precise interaction with a location if compared to the use of GPS only, since we can place the sensor exactly where we want the location i.e. a laboratory that is located at the seventh floor of a building, in the first room to the left. More specific missions that aligns with students needs can also be part of an location i.e. a mission from a laboratory that rewards the

player when a registration is fulfilled. This precision is not possible with only GPS, as it does not provide vertical positioning of a device. Other problem that can affect GPS accuracy is known as "GPS drift", where the positional information obtained from GPS satellites are affected mostly by the changing GPS satellite constellation patterns used by the guidance device to derive positional information [34].

The implementations described above only grasps the various existing possibilities when it comes to the union of the virtual game world with the real player world, but clarifies the potential present in the assimilation of IoT devices in a context of pervasive games. More missions implementations are described in chapter 6, where we describe our public experiments.

Some elements of using IoT devices in games were observed and can be explored for increasing the experience of games, for boosting the potential of the integration of participatory sensing tasks and for enabling the addition of a larger variety of serious applications. These elements are hereby listed.

- The search for IoT devices can be guided by the game, allowing the search for devices in any desired location, with a capacity of positioning on very specific locations, such as specific rooms on buildings with several floors;
- The devices can be randomly found at chance, or through exploration that is not directed to the exact location of the device, but to an area that the device is within. This can boost player's experience as it promotes the exploration aspect of the pervasive game;
- Devices can be attached to moving objects or entities. This can give a very interesting dynamic to the interactions with the devices, and can be explored for numerous gameplay mechanics;
- The devices possess a short range communication technology, which guarantees that the person in contact with it is near the sensor location. It is also possible to limit the signal strength allowance for limiting even more the distance of communication. This trait guarantees the presence of players in a specific location with high precision, and can be used for participatory sensing tasks and serious applications. For the sensing tasks, we support that it is more likely that the data will be collected with precision if the player is already at the location of the collection, as the effort of reaching the location is already done;
- Similar to the last item, it is also possible to guarantee the presence of groups of players in a specific location of any type through the short range communication trait. Certain aspects of a pervasive game be benefited from this characteristic, such as group missions that needs at more than

one player to be activated, increasing the variety of experiences that a game can offer. This can also be used for any application that needs the grouping of people in one specific place, including serious applications and sensing tasks.

4

Development and Runtime Environment

In this chapter we will detail the elements that composes the development environment of this project. Moreover, a setup guide is provided in the appendix of this document to aid in the preparation of any related project that desires to integrate IoMT devices within mobile pervasive games through ContextNet middleware [25]. Other tools used for the development of the game will also be explained, such as LibGDX framework [39] and 2DMapBuilder map editor [29].

4.1

Android Studio

When dealing with large projects, it is definitely recommended to work with a Integrated Development Environment (IDE). On top of that, when developing applications for Android platforms, you will need specific development kits and the ability to compile for that platform. With that in mind, we opted for using the Android Studio IDE, which is one of the most recommended for developing android applications [44].

4.2

ContextNet Middleware

Project ContextNet [2] aims at provisioning context services for wide-and large-scale pervasive collaborative applications such as on-line monitoring or coordination of mobile entities' activities, and information sharing through social networks. These entities may be users of portable devices, such as smartphones, vehicles, or autonomic mobile robots. In the ContextNet project, communication and context distribution capabilities are implemented in the Scalable Data Distribution Layer (SDDL), while other services and extensions are built as software modules on top of this distribution layer. Together, these software modules forms the ContextNet middleware [25].

In the experimental game implementation, we took advantage of the ContextNet middleware not only to enable discovery and communication with IoMT devices, but also to create the game client-server architecture, through

ContextNet Scalable Data Distribution Layer (SDDL). All communications between the game client and the game server are achieved via the SDDL.

To be able to discover and connect with IoT devices, we used the Mobile Hub API that extends the ContextNet middleware. More specifically, we utilized one of M-Hub available services, called S2PA. The S2PA service is responsible for discovering and monitoring nearby M-OBJs (IoMT devices) that use the supported WPAN technologies. This service keeps a record of the current provided sensors/actuators (e.g. temperature, accelerometer, humidity, etc.) and publish the sensed information to all the components that require it [26].

Both server-client communications and the use of the S2PA service are illustrated in figure 4.1, which provides a better view of the game's communication functioning.

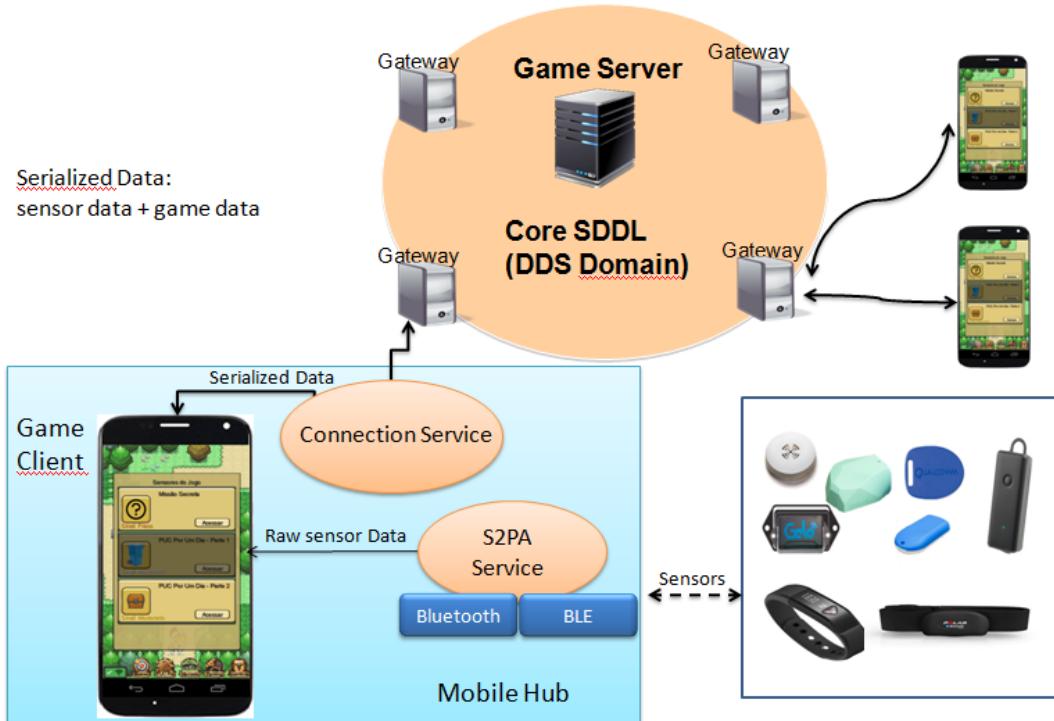


Figure 4.1: Experimental game client-server and sensor communications model

4.3 LibGDX Framework

LibGDX is a Java-based gaming development framework, with support for different platforms (Windows, Mac, Linux, Android, iOS, BlackBerry and HTML5), open source, fast and with various functionalities [39]. LibGDX was chosen for the development of the pervasive game on the Android platform, with the goal of facilitating development. It is a highly recommended frame-

work for Android game development [43], with almost five thousand games in its showcase gallery [42].

4.4

2DMapBuilder Map Editor

A tile-based game is a type of game in which the virtual world is composed of small areas, usually rectangular, called tiles. The creation of the world occurs by the joining of several tiles available for use, present in a group of tiles commonly called tileset or tilemap [40].

To facilitate the creation of scenarios for tile-based games, a tile-based map editor is usually recommended. This type of editor increases the efficiency of the design of the world. 2DMapBuilder (figure 4.2) is an editor in a very advanced state of development that turned out to be very helpful for the world design of this project. 2DMB has the purpose of assisting in the creation of the proposed game universe as it provides numerous editing tools [29].

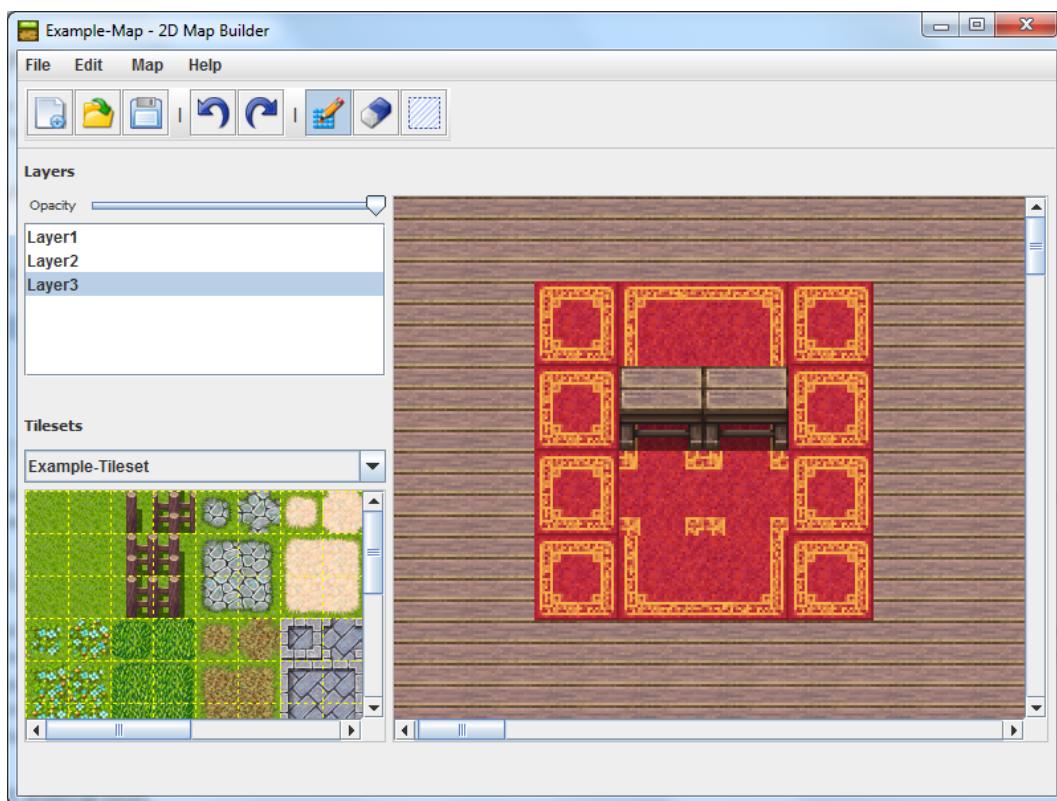


Figure 4.2: Example map in 2DMapBuilder map editor

It supports square shaped tile maps, tilesets importing and layers management. Layers define a level of the map, which can overlap or be overlapped by other layers. Each layer consists of tile groups that shape the different levels of the map. This way, layers separate the content of the map giving the ability to stack virtual objects in a 2D world e.g. a virtual table can be stacked on

top of a carpet that is on top of a wooden floor. A tileset can be of any image type and can contain one or many tiles. A tile has always a layer and tiles of inferior layers are rendered behind those of higher layers. In 2DMB, layers are automatically created and tiles are automatically set to their respective layer, but it is also possible to manage them manually.

To create your map, you'll have at your disposal several tools designed to help the efficiency of map creation. The primary tool for painting is the Brush Tool that allows you to paint a tile or a group of tiles into the map, accepting well-known modifiers of selection Ctrl-Click and Shift-Click, as well as drag-and-drop selection. The Eraser Tool will help you deleting single tiles of your map shifting automatically between layers. Finally, the Select Tool will give you the skill of cutting, copying, pasting and deleting a tile or a group of tiles in the selected layer from the map.

In order to give you more control, 2DMB provides a state mechanism that allows you to go back and forth in your map modifications, the famous undo and redo operations. A detailed manual on how to use 2DMB can be found in its documentation page [29].

To facilitate the parsing of the created map file, 2DMB's own file extension ".m2d" [41] was projected using XML notation and uses a simple formatting for the map data.

4.5 Gradle Build Tool

Gradle is an open-source build automation tool focused on flexibility and performance. It supports many major IDEs, including Android Studio, Eclipse, IntelliJ IDEA, Visual Studio 2017, and XCode [46].

Gradle is responsible for enabling the integration with all the necessary technologies, including the ContextNet middleware and the Mobile Hub component, as well as the LibGDX game framework. To use a build tool such as Gradle is not usually a hard task, but using it to integrate the framework LibGDX with other necessary libraries for the development of an IoT pervasive mobile game ended up being an arduous work. This is due to the multiple Gradle build files that the LibGDX framework generates, where each must be correctly configured for the correct compilation of the project. To diminish future efforts and avoid the headaches of the integration, we provide all working Gradle build files configured for this project in the appendix of this document, as well as a guide for using Gradle with the LibGDX framework in the A.4 section of the appendix.

5

PUCmon Development

This chapter details several parts of this project, in order to guide through the development process of the experimental game PUCmon and to better clarify the traits of the game, hopefully paving the road for similar projects.

5.1

Client-Server Architecture

To enable the online features of the game, a major part of the game's workload is placed in the game server. The game server is responsible for managing any data that can be of service for multiple clients, as well as storing game character's save data. Storing the save data of in-game characters into the server allows users to switch devices while still being able to use their characters. More details about the game server will be discussed ahead in this section.

5.1.1

Communication Architecture

ContextNet's SDDL layer provides the infrastructure to start developing applications capable of communication. There are plenty of tutorials in ContextNet's documentation [51] that will guide the initial implementation of client-server communication.

Once the base client and server implementations are fully working, meaning that the messages are being correctly sent and received, the desired communication architecture can be implemented in accordance to what the application requires, in order to facilitate further client-server operations.

5.1.1.1

Message Classification

To better organize the communication, the handling of messages was isolated from both client and server implementations. In addition to that, we used a type for every message sent, a classification that symbolizes the message content. Those types range from sign up messages to market messages (the market is an online feature of the game that allows players to buy and sell

virtual items to each other), and they help to standardize the structure of each message. Each type of message also carries a information regarding the desired action requested by the client. This helps to determine the needed operations to fulfill that message goals, since messages of the same type can have different intentions i.e. market message may be intended for item registration or item withdraw in the market.

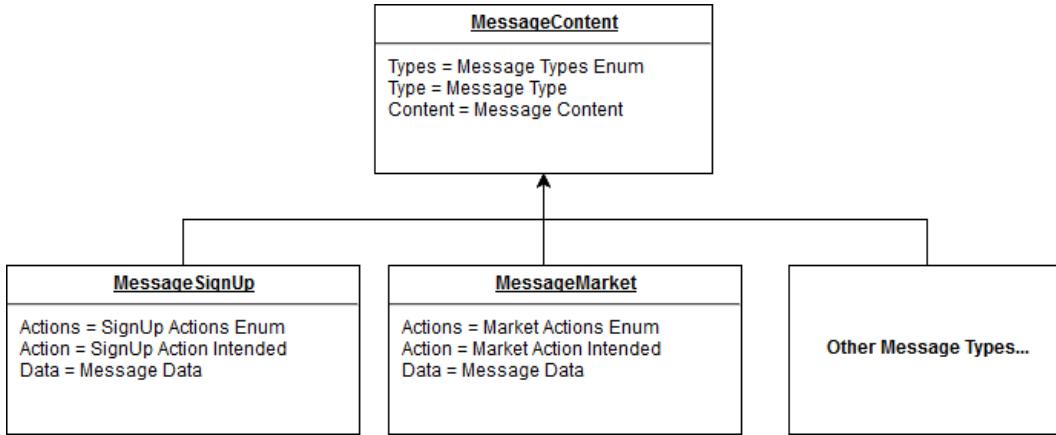


Figure 5.1: Messages classification architecture illustration

Figure 5.1 illustrates the message classification model. As can be seen in the model, different types of messages have similar structure, containing a list of possible actions for the type, as well as the current intended action for the message and the message data. To be able to send the message via the SDDL infrastructure, every message regardless of its type must be wrapped in the *Content* field of the *MessageContent* object. Notice that the *Content* is a field of the Java *Serializable* type, making every message also *Serializable*.

Finally, to be able to correctly unwrap the message content, the message type should be known. Thus, the *MessageContent* object also carries the list of possible message types as well as the type of the current message, enabling the understanding of each message delivered.

5.1.1.2

Message Distribution

Following the above-mentioned strategy, each message received by both client and server are sent to external handlers. The server handler is in charge of identifying each message's type and action, proceeding then to perform the required operations, including database manipulations and data retrieving. One detail not explicitly mentioned yet, is that each message also carries the information about its sender, a feature that is present in the SDDL infrastructure. This results in a easy way to reply received messages by the

server, as even with multiple clients requesting data from the server, there is no confusion to what client the message should be heading to.

The difficulty comes in another aspect of the communication: the distribution of the messages within the client application. More explicitly, there is a need to distribute the client's received messages to the respective module that prompted the server response message. The strategy adopted to distinguish each recipient was to simply embed the module's name within each message sent. That way, the distinction can be easily made, the only requisite is that the server includes this information on the reply.

With that in mind, we implemented a simple design that allows any class to easily communicate to the server in both directions, facilitating the development of the game's client-server operations. The whole scheme of the architecture is exemplified in figure 5.2.

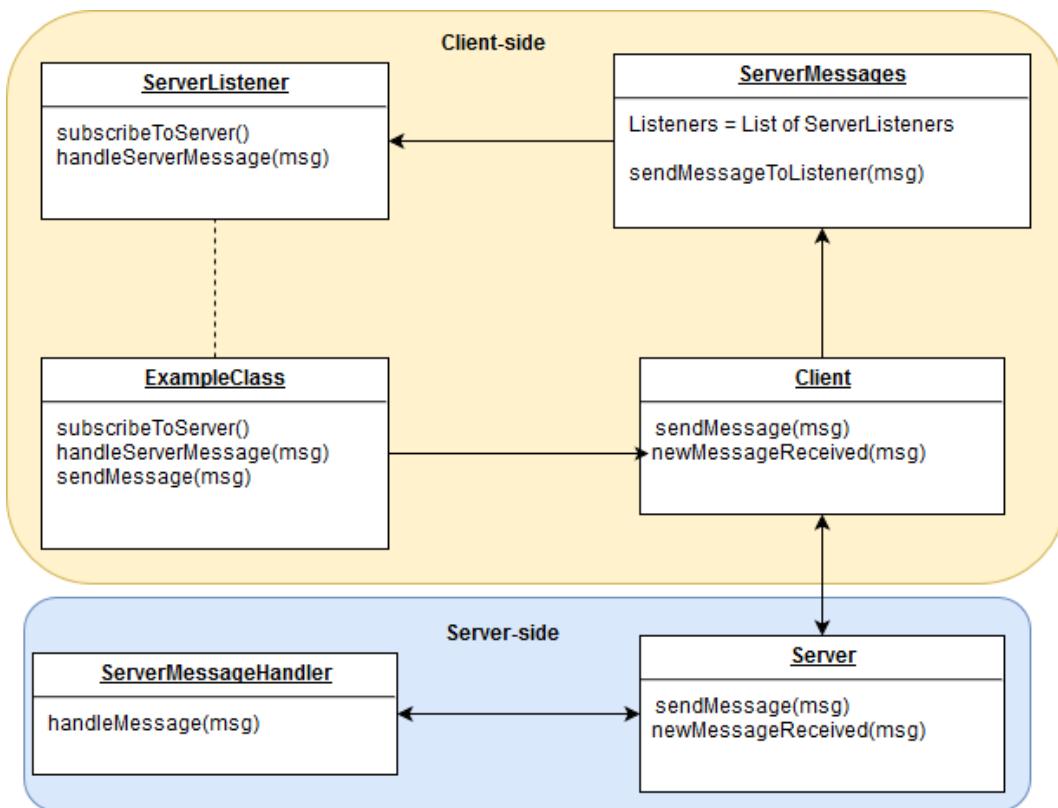


Figure 5.2: Message distribution architecture illustration

As represented in figure 5.2, a class can send messages to the server via the client module, but if the class intends to be able to receive server responses, it needs to implement the interface named *ServerListener*. The moment a class implements this interface, it becomes apt to be included in the list of server listeners that is present in the client's message handler, *ServerMessages*. To be included in the list of server listeners, the class must subscribe to it via the *subscribeToServer()* method. Once those requirements are met, for

every message received by the client, the class will be considered as a possible recipient of the message, and if the class name matches the information carried in the message, the method `handleServerMessage(msg)` of the class will be invoked, passing the received message as an argument. That way, each class object will handle received messages in the desired manner.

The explanations above aims to detail the design we implemented for facilitating client-server operations. The goal is to facilitate the development of future projects, but it does not mean that this is the only alternative. Surely enough, there are better improvements or even better designs to begin with, but since this architecture fulfilled our demands, we felt that it might be of use for future projects.

5.1.2

Database Architecture

The game's data is stored in the server-side, allowing users to play with their characters on any device, as their account data is retrieved from the server at login time. This means, that the game requires the user to have some internet connectivity. As a safeguard measure, if the user happens to lose that connectivity during the game, all new data regarding the game's offline mode, such as character evolution and item acquisition, are stored within the user device, enabling the player to safely play the game offline, without losing the gameplay actions and effects, but restricted of online features. The data in the mobile device is stored as key-value pairs using the Android `SharedPreferences` API. Once the connectivity is regained, the data is then synced with the server database.

5.1.2.1

Database Model

The game's database is responsible for storing not only account data, but also player's items, market entries, registered real-world (IoT) device sensors, and more. Figure 5.3 illustrates the game's database model. Also, the appendix contains the SQL schema file to act as a guide upon later projects.

The tables shown in figure 5.3 have the following purposes:

- *accounts*: Responsible for storing game accounts;
- *players*: Responsible for storing players of the game;
- *market*: Responsible for storing game market entries;
- *player_items*: Responsible for storing player's inventory items;
- *player_equipments*: Responsible for storing player's current equipment;

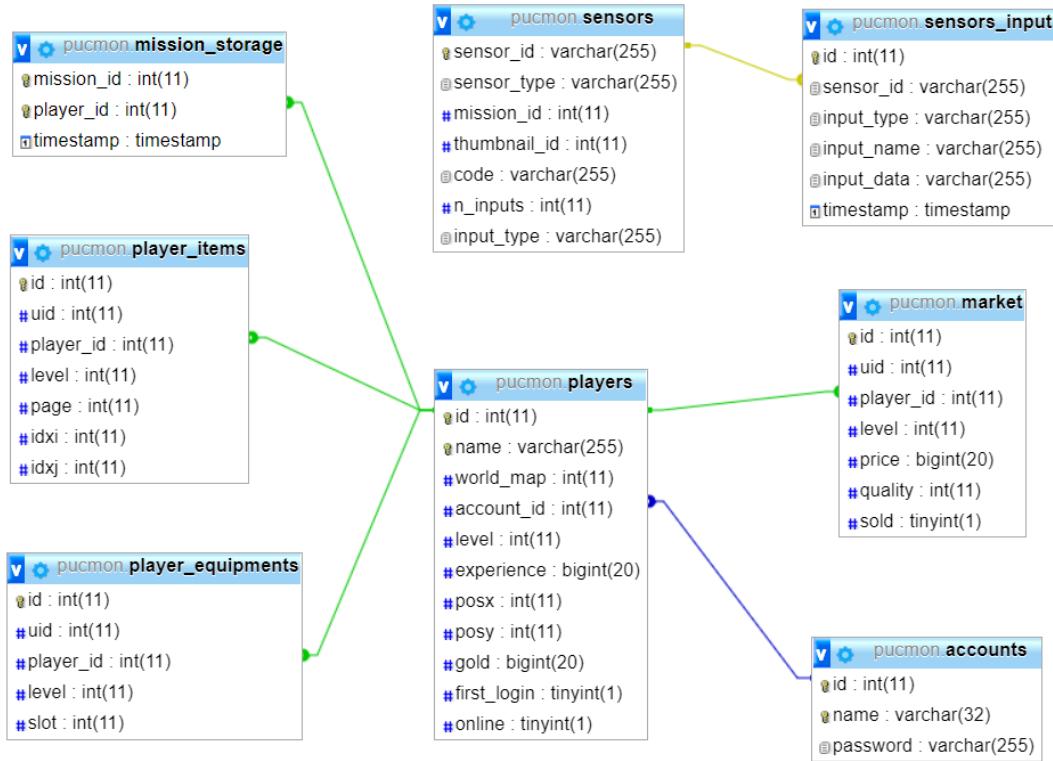


Figure 5.3: Database Model generated in phpMyAdmin

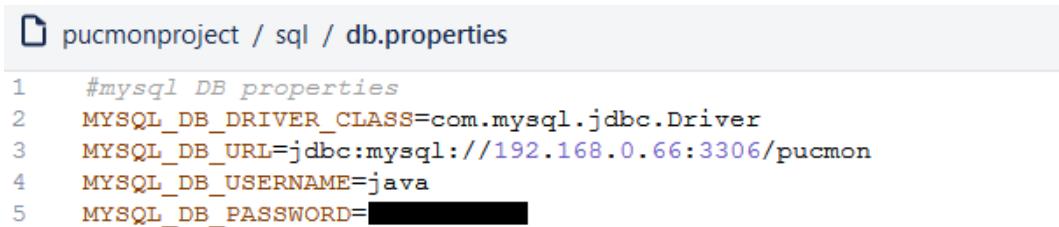
- *mission_storage*: Responsible for storing player's completed missions;
- *sensors*: Responsible for storing game's registered sensors;
- *sensors_input*: Responsible for storing input of game's registered sensors.

5.1.2.2 Database Integration

The Java Database Connectivity (JDBC) API provides universal data access from the Java programming language. Using the JDBC API, it is possible to access virtually any data source, from relational databases to spreadsheets and flat files [52].

The JDBC API was chosen to be the interface between the Java language and the database, since it is bundled within the Java Platform Standard Edition (Java SE) 8 release [52] and supports MySQL. It is fairly simple to use and should be no trouble in the development process.

Having said that, there are some helpful steps to organize the database interface. First, using a *properties* file to store connection information as shown in figure 5.4 is a nice way to manage the connection data. Also, creating a data source factory that produces the data source object with all needed data to connect to the database is also helpful, as it reduces duplicated code. Finally, have a separated helper for database operations.



```

pucmonproject / sql / db.properties
1  #mysql DB properties
2  MYSQL_DB_DRIVER_CLASS=com.mysql.jdbc.Driver
3  MYSQL_DB_URL=jdbc:mysql://192.168.0.66:3306/pucmon
4  MYSQL_DB_USERNAME=java
5  MYSQL_DB_PASSWORD=[REDACTED]

```

Figure 5.4: Database properties file example

In case there are some doubts about creating the data source factory, the appendix section contains the factory implementation used in this project.

5.1.3 Account Management

As discussed in previous section, the users must create accounts in order to be able to play the game. On account creation, they also create their characters, as shown in figure 5.5. Once the account and character are created, the game is fully available to the users.

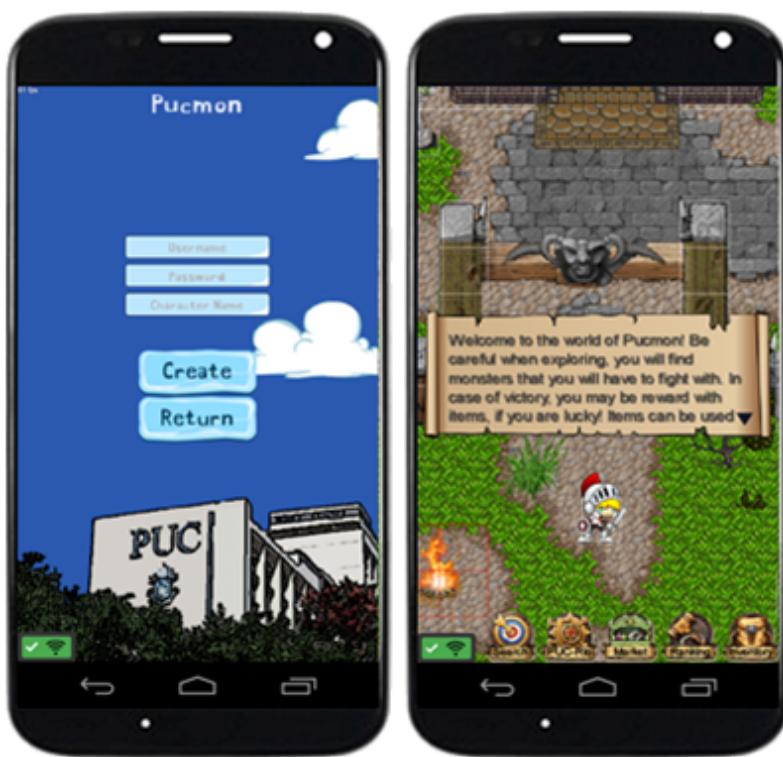


Figure 5.5: Account creation screens

There is one concern that needs to be addressed regarding the management of player's accounts. It is necessary to control the access to the game world in regards to players that are already on in the game. This means that a choice should be made, whether the new access should override the previous

one and drop the logged player, or the access of any player that is already logged should not be possible. The later was the path we choose.

In any of the cases, there is a need to have a control of the online players. One can argue that updating the online status of a player through log in and log off is enough, but if the players lose connectivity before logging off, the online status would not be changed. The work-around we did was having each online player send a tick within a time interval. If at any moment a player tick interval surpasses the stipulated limit, the online status changes to offline.

5.2 LibGDX Coordinate System

LibGDX framework has an extensive documentation that guides the development of applications that uses the framework [39, 47]. It is similar to many other game developing frameworks in several aspects, but there is a rather peculiar aspect of the framework that should be specifically explained, as it can cause some confusion - the LibGDX Coordinate System.

The most common usage of coordinates system in 2D games is the one called *y-down* (figure 5.6), which utilizes the top-left corner as the origin and the *y* grows in the down direction while the *x* grows in the right direction. It is also how most of the 2D graphic editors work. Unfortunately, that is not the case when it comes to libGDX. LibGDX coordinates system is tricky, as it varies depending on the coordinates space where it is applied. We will focus on the two main coordinates space used in this project: the touch coordinates and the screen coordinates.

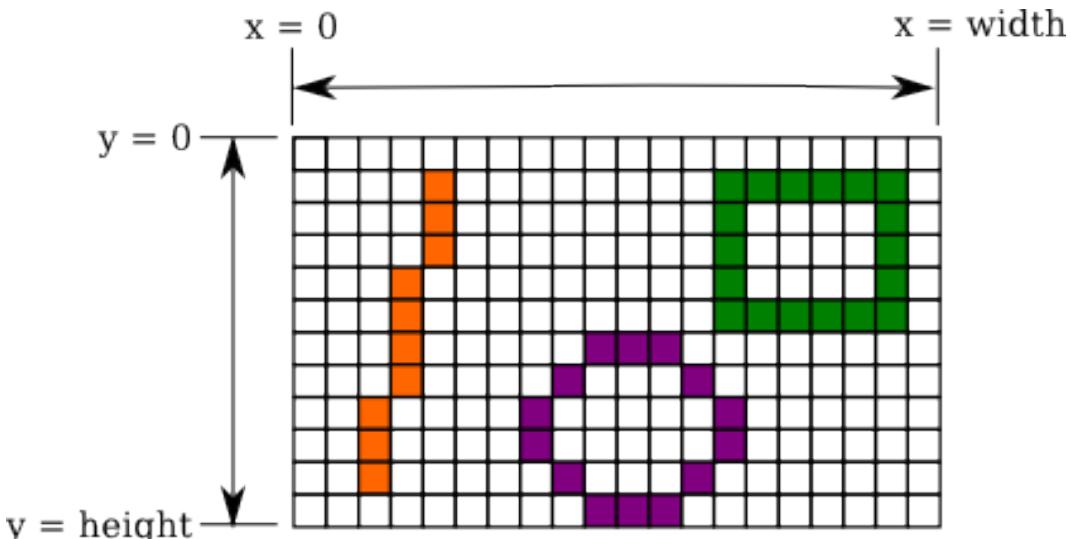


Figure 5.6: Y-down Coordinate System

5.2.1

Touch Coordinates

Whenever working with touch coordinates, the system used will be the *y-down*, with the dimensions of the physical screen in pixels. Typically, those coordinates tend to be converted to world coordinates, which represents our virtual world, because a considerable part of the interactions done with the screen aims to interact with the virtual world. This conversion process can be done through simple mathematical operations, usually based on camera's position and zoom.

Note that when interacting with fixed UI elements, there is no need for this conversion, as their position in screen is directly related to the touch coordinates, but another complication can occur, since the rendering of elements in the screen follows a different coordinate system by default, which will be discussed in the next section.

5.2.2

Screen Coordinates

The system used in screen coordinates is called *y-up*. The difference when comparing to the *y-down* is in the *y* axis, as the origin is now on the bottom-left corner and the *y* grows in the up direction. This may cause a mismatch that leads to future problems that can be avoided by standardizing the coordinates system. Luckily, this can be easily done through a simple operation with the *y* value. For instance, conversion from *y-up* to *y-down* would look similar to the following in the code:

```
y = Gdx.graphics.getHeight() - y;
```

5.3

Screen State Architecture

A game is usually composed of several different screens. Most of the time, the rendering of each screen is exclusive and separated from each other, meaning that at any given moment, only one game screen is being rendered. An approach that organizes the screen rendering of each menu with the same logic just described is effective and should be used on any application with similar characteristics. In this section, we will display our approach to the rendering of the game screens in our project, hoping that it might be of use to others who experience situations alike.

5.3.1

LibGDX Game Loop

Before we describe the design we implemented, there is an important aspect of this project that influenced in the decision for the pattern. LibGDX, much like other game frameworks, provides a game loop architecture that controls the execution of the game. The loop is composed of different callbacks that have distinct jobs and are executed by the framework at distinct times. These callbacks are only triggered in the main game module, which is the one that extends *ApplicationAdapter*. The most important callbacks that are part of libGDX framework are:

- *create()*: Called on application initialization only, should be used to implement initial configurations and load static resources;
- *render()*: Called every frame before rendering, should be used to update and implement what is to be rendered on the screen;
- *pause()*: Called when application is paused, should be used for operations related to the pausing of the game;
- *resume()*: Called when application is resumed from a paused state, should be used usually for the opposite operations of the *pause()* callback;
- *dispose()*: Called when the application is destroyed, should be used to dispose allocated resources;

There other important callbacks, such as screen touch callbacks that helps the control of user interactions, but we will focus on the game loop ones to better explain the implemented state architecture.

A libGDX application should follow the provided architecture in order to optimize the game development, and there is a way to expand it to create an ideal architecture for the game screens, which will be discussed in the next section.

5.3.2

State Design Pattern

The idea of the design is to treat each game screen as a state that is exclusive, much like a state machine. In order to be able to transition between each state with little effort, we used what is common to every state, the libGDX game loop. By creating an interface containing the game loop methods that act as a standard for each state, we are able to change the current state with a simple variable attribution, with the only requisite being the interface implementation.

There is one more detail we must not forget, the game loop callbacks are only called in the main game module, so we must delegate the execution of the game, which means that the main game module will act mainly as a switch for the different states of the game. To embed this idea in the libGDX game loop, we can create in the main game module a variable that holds the current state, which can be any object that implements the game state interface created. We initialize the variable with the initial state of the game, and once an event triggers the change of state, the main module simply needs to change the current state variable to the new state. Since the main module only delegates execution, there is no need to implement the game loop callbacks. Instead, it should only delegate the execution to the current state. Bear in mind that on the *create()* callback the main module should call every game states' *create()* method, to guarantee that every needed resource is initialized. The same goes for the callback *dispose()*, to make sure every resource is disposed. For the others callback, only delegating to the current state is enough, e.g.:

```
/*
 * Called when the Application is resumed from a paused state,
 * usually when it regains focus.
 */
@Override
public void resume() {
    // pass the resume control for the current state being
    // rendered
    currentState.resume();
}
```

Figure 5.7 gives an overview of the architecture of states in order to aid in the comprehension of the implementation for the design in a libGDX game context. It shows the main module having the *currentState* variable that stores the current state of *gameState* type, the *changeState()* method that changes between possible states of the game, and the callbacks that only delegate execution. In the case of the *create()* and the *dispose()* methods, the implementation is not visible for its extensive nature, as both should call every game states, instead of only the current state. The interface *GameState* is also displayed and several states that implements the interface are exemplified in the overview.

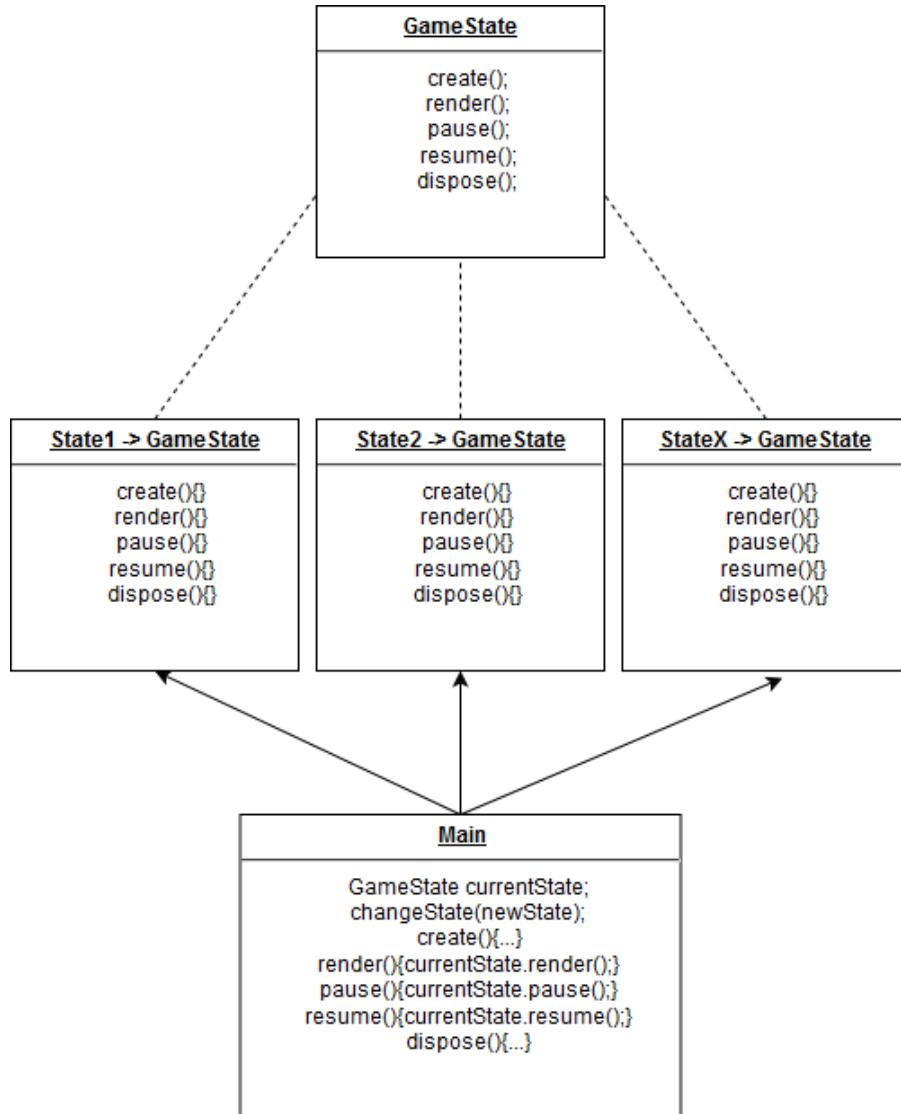


Figure 5.7: State architecture overview

5.4

Multiple Language Support

It is almost a must for every game in this moment in time to support multiple languages in order to reach for larger audiences. LibGDX provides a mechanism to help the organization and the implementation of multiple languages in a game, through the `I18NBundle` class.

A `I18NBundle` provides Locale-specific resources loaded from property files. A bundle contains a number of named resources, whose names and values are `Strings`. A bundle may have a parent bundle, and when a resource is not found in a bundle, the parent bundle is searched for the resource [53].

The property files should be related to each desired language containing game strings in the respective idiom. Figure 5.8 shows an excerpt of the English and the Portuguese property file used in this project.

android / assets / language / language_en_US.properties	android / assets / language / language_pt_BR.properties
<pre> 1 # general properties 2 playStr = Play 3 langStr = en-US 4 hudInfo = {0} Lv. {1} 5 levelInfo = Level {0} 6 inventoryStr = Inventory 7 profileStr = Profile 8 usernameStr = Username 9 passwordStr = Password 10 characterName = Character Name 11 newUserStr = Sign Up 12 createStr = Create 13 returnStr = Return 14 goldStr = Gold 15 # ranking properties 16 positionStr = Position 17 nameStr = Name 18 levelStr = Level 19 charRankInfo = Your Position: {0}</pre>	<pre> 1 # general properties 2 playStr = Jogar 3 langStr = pt-BR 4 hudInfo = {0} Nv. {1} 5 levelInfo = N\u00EDvel {0} 6 inventoryStr = Invent\u00EDrio 7 profileStr = Perfil 8 usernameStr = Utilizador 9 passwordStr = Senha 10 characterName = Nome do Personagem 11 newUserStr = Cadastre-se 12 createStr = Criar 13 returnStr = Voltar 14 goldStr = Ouro 15 # ranking properties 16 positionStr = Posi\u00E7\u00E3o 17 nameStr = Nome 18 levelStr = N\u00EDvel 19 charRankInfo = Sua Posi\u00E7\u00E3o: {0}</pre>

Figure 5.8: English and Portuguese property files

Once the properties are created, setting a language and using it is very simple. You can set the language at any time in your code, so it is recommended to provide the user an option to choose the language in a preferences menu. The code below exemplifies the usage of the I18NBundle to clarify how it is done.

```

// language bundle
I18NBundle langBundle;
FileHandle langFileHandle;

// language bundle initial creation
langFileHandle = Gdx.files.internal("language/language");
// creates locale with example language and country
Locale locale = new Locale("pt", "BR");
// creates language bundle with locale
langBundle = I18NBundle.createBundle(langFileHandle, locale);

// from this point on, any key word retrieved from langBundle
// is within the desired language

// example text 1 - gets the string in the current language
String play = langBundle.get("playStr");

// example text 2 - gets the string in the current language
// formatting with
// the level information in the first argument of the string
// symbolized by {0}
String levelInfo = langBundle.format("levelInfo", "10");
```

5.5 Configuration File

A good idea when dealing with creation of game applications it to externalize certain configurations to be able to tune it. This can be done in another Java file or in a external resource file. In the later, no compilation is needed to apply changes. The code beneath is an extract of this project configuration file, showing some configurations that are very sensible to changes, as its regulation is largely due to experimentation.

```
*****
 * Reward rate constants *
 ****
expRate = 1.0;           // exp rate global modifier
goldRate = 1.0;          // gold rate global modifier
dropRate = 1.0;          // drop rate global modifier
```

The rate of experience, gold and drop are always being tuned to achieve the proper difficulty desired, and this is not an easy adjust. It is vital that these values are easy to change. This situation applies to diverse others parameters of games. For those cases, having a configuration file, or even multiple ones, can mean an improvement of the game's development efficiency.

5.6 Text System

It is very common in the RPG genre to have a form of text system that serves as a way to communicate with the player. These systems allows texts to be displayed in a paced manner and appears on certain events in a game such as NPCs conversations and battle information.

With that in mind, we designed a text system that has the ability to take large portions of texts and automatically break down into smaller sections that fits into the system background, that is displayed separately until the whole text is finished. The displayed texts are compatible with the specified language, as the source for the texts are placed in the property files of each language. For each section displayed, there is an animation that prints letter by letter at a fast rate, but if the player wants to speed up the animation, all that is needed is a simple touch in the screen. When creating texts for the system, there is a possibility to anchor it in different positions depending on what is best for each specific tests i.e. top, middle, bottom.

We recommend that any similar text systems in game should have similar characteristics, including an animation that smooths the text display and the

option to speed it up, as long texts with slow paced animations may hurt the player experience a bit. It is also important to block all game actions that are not related to the text system while a text is being displayed, leaving the player in a calm state of mind to read each text as nothing is interfering it. Finally, it is also a nice practice to have a visual feedback representing that there is more text ahead such as an animated arrow, which is the case of this project's system.

5.7 Sound Effects

Sound effects are a very important trait in any game, and they can elevate the player's experience to a whole new level if done correctly. From BGMs to battle effects, nowadays the sound design in games is very valuable and is done by professionals of the field.

Although we understand the valor of the sound design in a game, we used copyright-free public sound effects to aid in the sound setting of the game, since it is an experimental game. But, as a token of our appreciation for game's compositions, the game's main themes were created from the scratch using a free software (figure 5.9) that lets you compose tracks that resembles the 8-bit era of games, which fits the universe of the experimental game. We intended to express the importance of sound design in increasing the experience of a game.

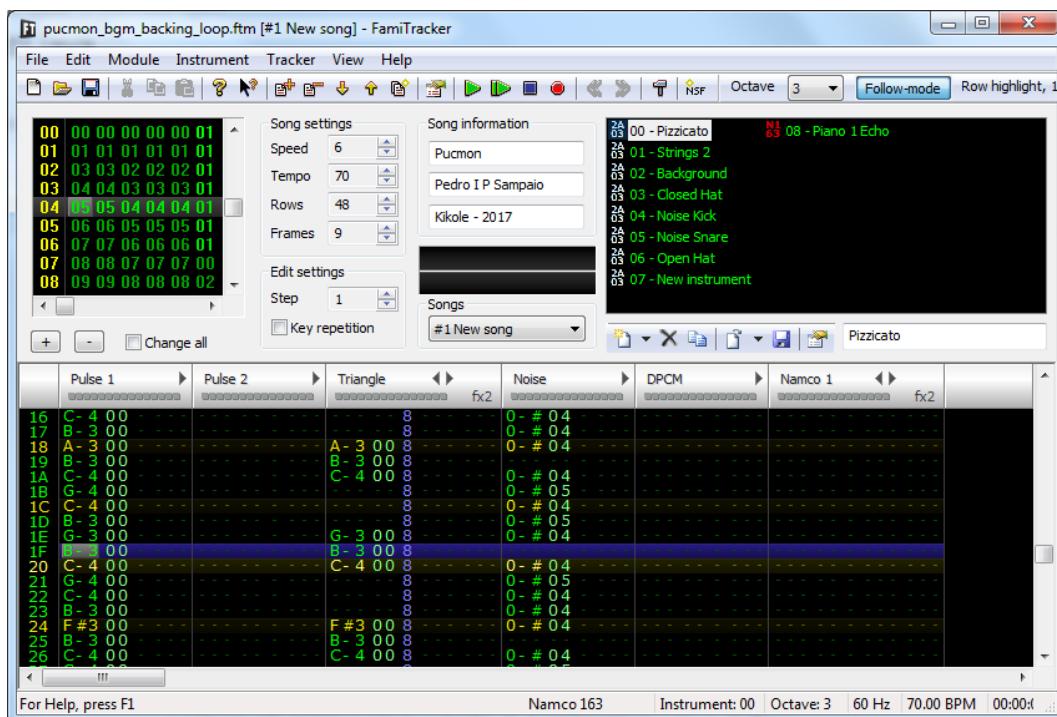


Figure 5.9: Game's BGM in FamiTracker software

While we cannot contribute as much as we want when it comes to the creativeness behind the composing of game sound effects, we can recommend a way of dealing with the implementation of them in the game.

Implementing the sounds using libGDX is not a hard task, as it has all the means to load a play audio files in several formats, supporting MP3, OGG and WAV files [54]. In this project, we used familiar data structures to create a sound bank that acts as a sound station in which it is possible to choose a sound or theme to be played at any moment by its defined name tag. The code beneath serves as an example of this idea implementation.

```
// enum with all available sound effects
public enum Effect {Hit_Enemy, Critical_Hit_1}

// station containing all sounds to be played
private HashMap<Effect, Sound> stationSFX;

// load sounds
Sound sfxCritHit1 =
    Gdx.audio.newSound(Gdx.files.internal("sfx/Critical_Hit_1.wav"));
Sound sfxHitEnemy =
    Gdx.audio.newSound(Gdx.files.internal("sfx/Hit_Enemy.ogg"));

// adds sounds to station attached to effects key
stationSFX.put(Effect.Critical_Hit_1, sfxCritHit1);
stationSFX.put(Effect.Hit_Enemy, sfxHitEnemy);

// plays sound effect at volume 1.0f
stationSFX.get(Effect.Hit_Enemy).play(1.0f);
```

5.8 Map

Prior to this section we defined the experimental game as a 2D RPG style. In those type of games, tile-based and top-down maps are the most common, and it is also the direction that we took for the game. Knowing that, 2DMB was developed to attend those characteristics, resulting in a software capable of designing, editing and creating an infinity of different 2D tile-based maps. In this section, we intend to guide through some of the important steps to use a 2DMB map in an optimized manner.

5.8.1

2DMapBuilder Integration

Integrating the 2DMB produced map file of *M2D* format is no hard task, since it is composed of *XML* notations, which has been extensively used and is not new by any means. Java supports more than one method for parsing XML documents out of the box. In particular, the approach that uses *DOM* documents was used as it is fairly simple. The code below shows a section of a *M2D* file produced in 2DMB, and as it is possible to see, it is indeed composed by *XML* notations.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<map>
<mapwidth>32</mapwidth>
<mapheight>32</mapheight>
<tilesize>32</tilesize>
<tileset>
<name>tileset_ex</name>
<source>tileset_ex.png</source>
<firstid>1</firstid>
<tilecount>192</tilecount>
<tilesize>32</tilesize>
</tileset>
<layer>
1,1,1,2...
1,8,7,2...
1,5,5,2...
.....
</layer>
```

Most of the data stored in the map files are easy to retrieve, but there are some details that deserves a careful look. There can be multiple tilesets used in a map. Also, one of the features of 2DMB is to let you compose the map in up to four different layers, each one acting as a z-index for the positioning of tiles and objects. In consequence of that, the map file will have repeated *XML* tags for those data. Luckily, it is not difficult to deal with this situation using the above mentioned java parser. The code ahead exemplifies the process of retrieving the tileset data, which should be the data that has more steps to obtain.

```
// the file containing the information to be loaded
InputStream fXmlFile = Gdx.files.internal("maps/" + map + ".m2d").read();
```

```

DocumentBuilderFactory dbFactory =
    DocumentBuilderFactory.newInstance();
DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
Document doc = dBuilder.parse(fXmlFile);

//optional, but recommended
// reduction of redundancies
doc.getDocumentElement().normalize();

// list of tileset nodes
NodeList tsList = doc.getElementsByTagName("tileset");

// iterates through list of tilesets nodes to gather information
for (int i = 0; i < tsList.getLength(); i++) {

    Node tsNode = tsList.item(i); // the current tileset node

    // gets information only if node is of element type
    if (tsNode.getNodeType() == Node.ELEMENT_NODE) {

        // the current element of tileset list
        Element tsElem = (Element) tsNode;
        String tsName =
            tsElem.getElementsByTagName("name").item(0).getTextContent();
        String tsSource =
            tsElem.getElementsByTagName("source").item(0).getTextContent();
    }
}

```

Once the data is correctly retrieved, the programmer may organize it in the desired data structures. Some data have a more obvious data structure match, such as the tiles of each layer of the map, that should be stored in a bi-dimensional array forming a 2D grid. Apart from that, the best way one knows to store the remaining data can be utilized with probably no major setbacks.

5.8.2 Optimized Map Rendering

Granted that the map tile data is stored in grids, or bi-dimensional arrays, a very optimized tactic of rendering can be used. It involves the current camera position, as well as the camera's viewport width and height, but there is no

problem to discussing it prior to the game's camera section, since it is the algorithm that matters.

The idea is to perform a clipping in the render in order to draw only the needed tiles. Let's assume our world grid is composed of 1024x1024 tiles, but our camera can only display a grid size of 16x16, it would be a huge waste of resources to always display the whole world, since the player can only see a small portion of it. Any kind of clipping is recommended, as it will certainly improve by miles the size capacity of the world. Our approach involves performing the clipping before rendering, as we clip the grid prior to rendering to match exactly what the player is able to see at any given moment. The following code express the algorithm that is used in this project.

```
// gets frontiers of displayed tiles of map grid to minimize cost
int first_tile_x = (int) Math.floor(cam.x / tileSize);
int first_tile_y = (int) Math.floor(cam.y / tileSize);
int offset_x = (int) Math.floor(cam.x % tileSize);
int offset_y = (int) Math.floor(cam.y % tileSize);
int n_tile_x = (int) Math.ceil(cam.viewportWidth / tileSize) + 2;
int n_tile_y = (int) Math.ceil(cam.viewportHeight / tileSize) + 2;
// draws each tile
for (int i = 0; i < n_tile_y; i++) {
    for (int j = 0; j < n_tile_x; j++) {
        // gets tile that will be drawn
        Tile tile = mapa[first_tile_y + i][first_tile_x + j]
        // do the necessary operations to draw the tile...
        // draws current tile
        batch.draw(tile.tex, (j * tileSize) - offset_x, (i *
            tileSize) - offset_y);
    }
}
```

From the code above, we can see the clip being made based on the camera's position and dimensions. The first tile that is indeed within camera's display is determined, as well as the number of tiles that fits the camera display. Bear in mind that the number of tiles is extended a bit to guarantee a smooth transition between frames in a way that does not affect the performance. Since the camera's movement is based on pixels and not on tiles, it will happen more than often moments where the camera is not exactly at the beginning of a tile, so we must also determine the current offset of the camera to be able to properly render the tiles. Once these calculations are done, we are able to draw every tile that currently is within camera's boundaries, as shown in the

code above, guaranteeing an optimized rendering through a clipping operation that has very low performance costs.

5.9 Game's Camera

As previously stated, having a camera in the game will help control what portion of the world is displayed to the player at any given moment, as illustrated in figure 5.10. It is most vital for games in which the world surpasses the screen limits. Each type of game have different camera implementations that fits better to their gameplay. We will be discussing the camera implementation for this project's experimental game, a 2D top-down RPG. With some tweaks, the camera implemented can also be adapted for other game styles such as 2D platformers.

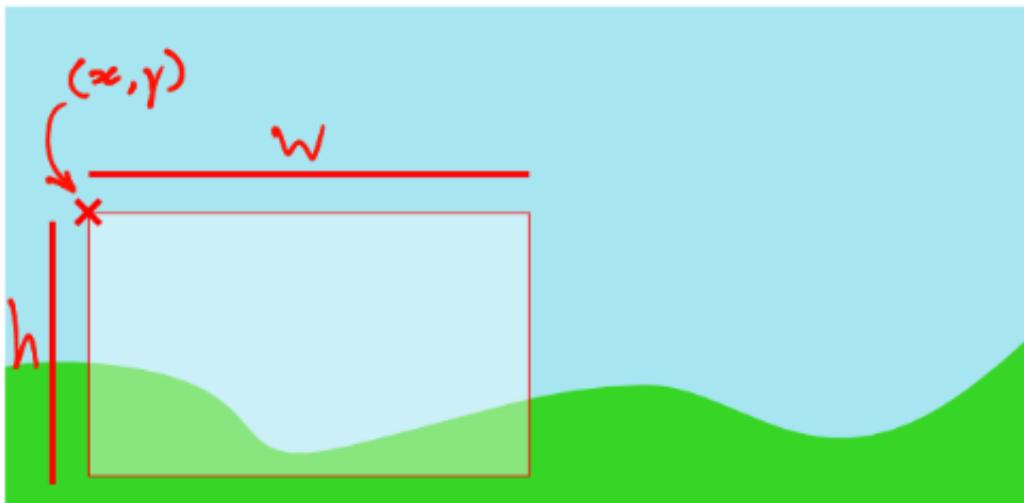


Figure 5.10: 2D game camera illustration

Figure 5.10 illustrates the basic information we need regarding the game's 2D camera, which includes the position of the camera in the world, as well as the camera dimensions.

The definition of the camera's dimension, also known as camera viewport, is a preference configuration, as it dictates the size of the world portion that will be displayed for the player. Having it set for a value that is too small or too large may affect negatively in the game's experience, so it must be carefully configured. Note that the camera's dimension may directly impacts the camera's zoom, as most cameras do stretch the displayed content to fit the device screen, leaving no unused pixel.

To define the camera's position we must first understand how the game camera should function in the game world. In the proposed game's style, the

camera is always aiming to follow the player and to be positioned in a way that makes the player be displayed at the center of the screen, so the definition of its position is directly related to the player's current position. One detail that must be considered is that the camera must not surpass the game world boundaries, so there needs to be a limitation regarding the camera's position, blocking it from displaying anything that is further world limits.

With that in mind, the resulting algorithm that implements the camera in the described way ends up being fairly simple. The steps to implement it revolves around calculating the camera's position based on player's current position, centering the player on the screen, and clamping the position in order to avoid world limits. The code below is a representation of these step.

```
// calculates max camera x and y based on the world size
// by calculating the world size in pixels (e.g. mapSizeX * tileSize)
// and subtracting the camera viewport size
max_x = (mapSizeX * tileSize) - getViewportSize().x;
max_y = (mapSizeY * tileSize) - getViewportSize().y;
// calculates camera position based on target's (player)
// position centering the target on the screen
position = new Vector2(target.getPosition().x -
    getViewportSize().x/2,
    target.getPosition().y -
    getViewportSize().y/2);

// clamp values avoiding going out of world boundaries
// using inferior limit zero and the calculated superior limits
position.x = MathUtils.clamp(position.x, 0, max_x);
position.y = MathUtils.clamp(position.y, 0, max_y);
```

How the map will use the camera's position to properly render the game is displayed in the previous section code example. Even though the camera's implementation of this style is not hard at all, if preferred libGDX provides its own implementation that can be used to create the game's 2D camera, through the *OrthographicCamera* class [55].

5.10

Animation Architecture

The technique of animation creates an illusion of movement through successive drawings that are showed in sequence. This technique is used in diverse media formats, including games. In this game, each animated object is a sequence of 2D images, usually named *sprites*, as exemplified in figure 5.11.

To produce the animation effect of an object in the game, we must define the sequence of sprites that composes the animation and control the alternation of sprites.



Figure 5.11: 2D sprites example

Usually the sprites are collectively stored in one image, as shown in figure 5.11. The image that holds collection of sprites are called *spritesheets*. This practice reduces the performance cost greatly by reducing the amount of needed resource loading. Most game frameworks provides the tools for cutting regions of loaded images, and we can use it to define the region that represents the sprite that we want to render. LibGDX does this through the class *TextureRegion*, which makes the job of rendering sprite sequences from spritesheets pretty simple and efficient.

The steps for producing the animation effect are very straightforward, and can be represented by these operations:

1. Cut the region of the current sprite of the animation;
2. Render the the current sprite;
3. Wait for the animation interval time;
4. Update the current animation sprite with the next one;
5. Repeat steps 1 to 4.

A common practice in games is to loop the animation sequence, meaning that when the render of the last sprite is finished, the sequence goes back to the first sprite to repeat the cycle. This is due to the nature of games, in

which players decides the actions of the animated characters and how long the action will last. It also reduces greatly the amount of needed sprites in a sequence of animation, as we only need the minimal quantity to create the desired animation cycled effect.

If the implementation of the animation effect is no trouble at all, to organize the enormous amount of different animations from different entities that exists in the game is the opposite. Usually there are multiple animation sequences for each game animated entity, having actions that changes between the available animation sequences e.g. idle animation of a character changes to the walk animation when the character starts moving.

To aid in the implementation of the animated entities in the experimental game, we created a structure based on finite state machines that controls each animated entity animations. The idea is that each state of the machine represents an entity animation and each transition represents the action that changes the entity current animation. Figure 5.12 shows a representation of the state machine of an animated entity that can attack, defend, walk or just be idle.

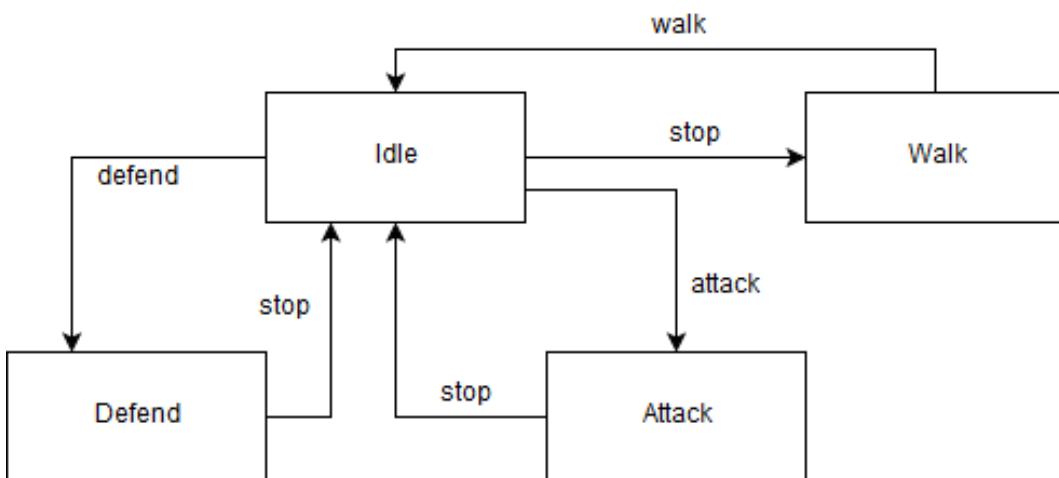


Figure 5.12: Animated entity state machine

The architecture implemented did not affect deeply the development efforts and has enabled an efficient and organized way to deal with the different animated entities of the game. We recommend that this approach should be considered specially if there are many animations present in the game.

5.11 Collision Detection

A game without any type of collision is extremely rare, as it is more than normal in a game to happen overlaps between its entities. Numerous

techniques exists with the goal of detecting physical collision, some of them are more precise while others are more efficient. In a 2D world like ours, the most common technique involves defining a rectangle shaped area for the entities of the world, representing their physical shape. Those rectangles are usually called hit boxes, and any time a hit box overlaps another one a collision happens. This is a very efficient strategy since the algorithm to check if two rectangles overlaps is very simple, as shown in the code below.

```
// x1,y1,w1,h1,x2,y2,w2 and h2 represents the origins
// and dimensions of the two rectangles
if x1 < x2+w2 and x2 < x1+w1 and y1 < y2+h2 and y2 < y1+h1
    collision = true
else
    collision = false
```

It is very common to already have a method to detect the overlap of two rectangles in a game framework, and libGDX provides this check through the method *overlap*. As we've seen, detecting the collision of entities in the game through the hit box method is very simple and efficient, but the difficult arises on organizing and structuring the collision system to provide a practical usage while also maintaining the efficiency. Our approach to deal with this situation is discussed ahead.

5.11.1 Collision Architecture

Taking inspiration from *Unity*, one of the most popular game engines as of today, we implemented an architecture that allows any entity of the game to have its own collider (hit box) and to deal with collisions with ease. The steps needed to take advantage of the system are minimal and the benefits are evidently. Figure 5.13 provides a superficial illustration of the system and its components.

As shown in figure 5.13, to use the collision system an entity of the game must implement the *Collision* interface, which contains the following callback declarations:

- *onColliderEnter*: Callback to deal with physical collision;
- *onTriggerEnter*: Callback called at the first frame a trigger collision happens;
- *onTriggerStay*: Callback called on subsequent frames when trigger collision is still happening;

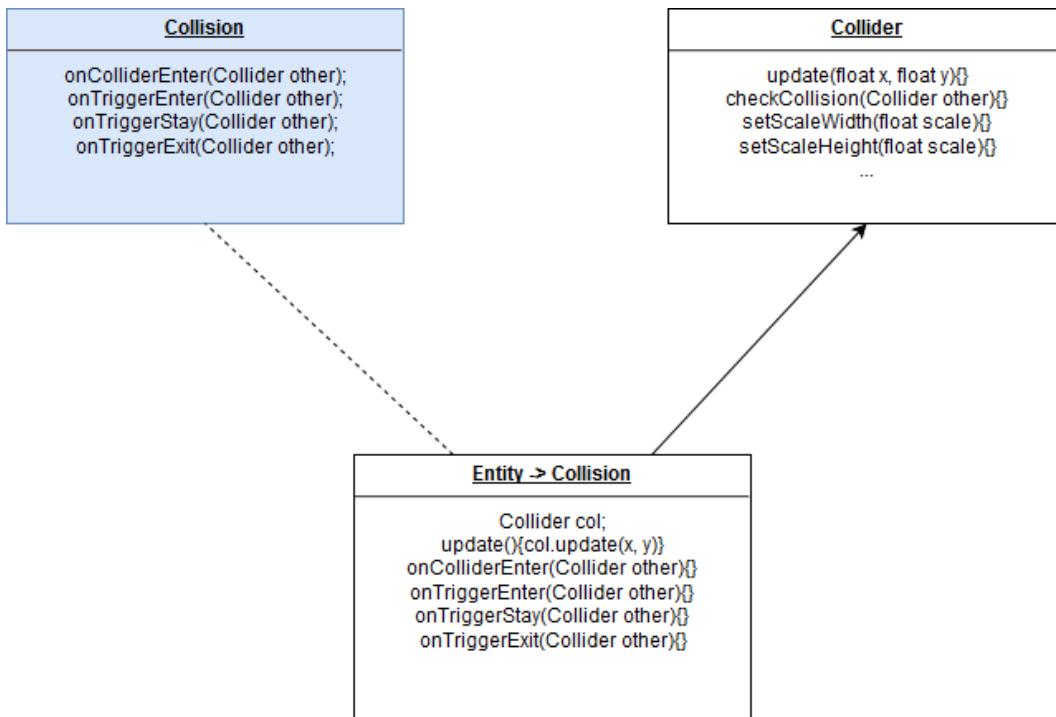


Figure 5.13: Collision system illustration

- *onTriggerExit*: Callback called when trigger collision stops happening.

Note that the difference of physical and trigger colliders is that physical colliders will automatically blocks movement when collision happens, while trigger colliders will not block the movement as it will only trigger the desired action when collision happens. This distinction is very useful, as there are numerous situations where we just want to detect the collision to trigger some effect e.g. player steps into a field that is on fire and proceeds to lose damage while on it. For this project, we felt a bigger necessity of having the separation in the callbacks for trigger collision. If wanted, physical collision callbacks can also be divided the same way as the trigger collision was.

One thing that happens that can not be ignored is that the entity move throughout the world. In this case, its collider should also move accordingly. So to make sure that everything is in sync, the collider *update* method updates the collider position appropriately, and should be used always for the case of moving entities.

The benefits of the system not only rests on the ability to create, scale, transform and attach colliders into any entity of the game, but also on the facility to deal with collisions through its different callbacks. For its practicality and usefulness, we recommend the described system for any type of game.

5.11.2

Collision Mapping

To aid in the mapping of the collision in the map, the newer version of 2DMB comes with a tool that allows to pinpoint fields with trigger or physical colliders. Figure 5.14 shows the usage of the 2DMB collider tool.

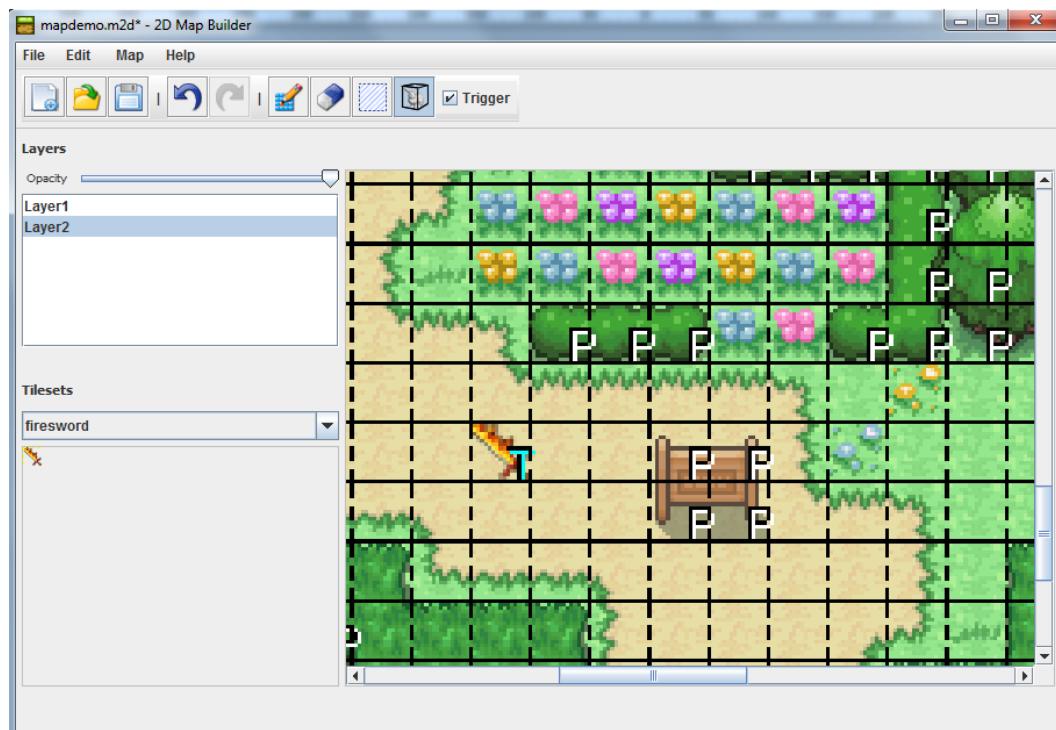


Figure 5.14: 2DMB collision tool usage

It is a very simple collision mapping tool, but can be of service for numerous types of games. The result of the mapping tool is stored in a collision mask file of *.col* format. It is basic mask of the map regarding the collision data of every field that belongs to the map. The excerpt below serves as an example of a portion of a collision mask file.

```
1,1,2,1,1,1,1,0,1,0,0,2
0,0,0,0,0,0,0,0,1,0,2
0,0,0,0,0,0,0,0,0,0,0
```

Note that *0* represents fields with no colliders, *1* represent fields with physical colliders and *2* represent fields with trigger colliders. This data can easily be used to define the map colliders, facilitating the implementation of the collision system.

5.11.3

Optimized Collision Check

One thing it is important to keep in mind when dealing with collision in games is that efficiency is a key word. Games differs from simulations where the precision is the most important trait. Even though using hit boxes as physical corpses for the entities is very efficient on its own, we must still be careful with the efficiency of our collision check. This is because at every frame in the game there will be an enormous amount of collision checks, specially if no optimization is applied. This behaviour is due to the fact that moving entities must ensure that they are not colliding with another physical object or a trigger object at every movement.

The naive approach to this problem is to check at every movement of every moving entities all the possible collisions. This will possibly grow into a costly approach and limit the scalability of the world. Our advice is to focus only at nearby fields, in areas surrounding moving entities. This is a very obvious reasoning, as it is clearly not necessary to check collision with a field that isn't even within visual reach. By limiting the collision check in this manner, the costs will greatly reduce resulting in a more flexible world building. One more thing that can be done that will be of great help depending on the type of game is to focus only in what is within player visual reach. Basically, if the player is not visualizing the entity, there is no need to worry about its collisions.

5.12

Character

A character in the experimental game is assigned to each player that creates an account to play the game, where they can choose the desired name for their characters. With their characters, players can explore the world moving freely in the existent maps, while battling with encountered foes and getting stronger throughout the journey. In section we will discuss some important details about the characters in the game.

5.12.1

Character's Level System

One of the most important traits of any RPG games is the evolution of characters. As they undergo through numerous challenges, they emerge stronger as more victories are achieved. To register their evolution, a numeric value is used. This value, usually called level, represents the current amount of

evolution a character has achieved. Players feel motivated to get higher levels as they also represent an increase in the character strength.

The rate of the progression is a choice that designers make based on their visions of the game. It is not an easy tune as it is adjusted throughout the whole development process. In this game, we wanted a infinite progression system that allows players to evolve their characters without any limitation. To achieve that, we used a mathematical function that gives the amount of needed experience points for the level entry. Choosing the correct mathematical function and its constants may be some trouble, and will depend on how the game progression will work. In our case, instead of a linear progression, we opted for an exponential progression. We intended to smooth the game difficulty curve, as the lower levels are easier to achieve, while the higher levels are harder. The following formula is from the current mathematical function used. Bear in mind that we are constantly tuning the formula.

$$e^{((x-a)/b)-c}/d$$

Figure 5.15 shows an illustration of the progression through the function chart, where the constants used are -26 for a , 6.2 for b , 78 for c and 5 for d . The exponential growth is very accentuated, but it is possible to tweak it through the constants. You may also want to consider other mathematical functions to achieve the desired progression curve, but we recommend using these types of functions instead of manually inputting the needed experience points for every level, as the latter approach would be very limited causing unnecessary development efforts.

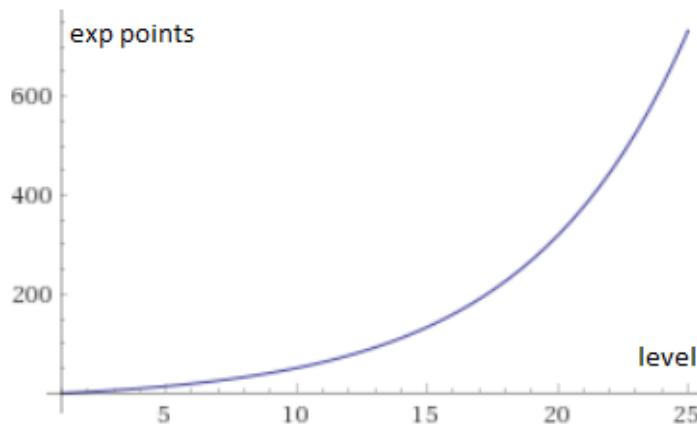


Figure 5.15: Level progress chart illustration

5.12.2

Character's Attributes

In the experimental game, there are many factors that affect the character overall strength. The fundamental one is the level progression. For each new level a character achieves, the stronger the character gets. Equipment, which will be discussed further in this document, is also a very important factor.

The character strength involves traits that are shaped in a variety of forms. Figure 5.16 shows the character attributes, which are the definers of the overall strength. Most of these attributes are very common in RPG games, such as attack power and defense rate.

 Max HP	350
 Experience	70,61%
 Attack	321
 Defense	160
 Auto Speed	7,06
 Auto Attack	233
 % Critical	41,19%
 Critical	x 18,88

Figure 5.16: Character attributes example

The attributes system was designed to be portable and easily attached to any entity of the game. This facilitate the definition of other entities that can engage in battles with the player, such as the enemies, which will be discussed further ahead.

5.12.3

Character's Inventory

For a RPG game in which item and equipment collecting is a feature, it is almost indispensable to have a inventory screen that helps the management of gathered items. To help players of the game managing their items, we developed a inventory screen that allows players to store a decent amount of items that are organized in six different pages. From the inventory screen is possible to swap items positions through different pages, and equip the equipment into their respective slots. Figure 5.17 shows the character inventory with many items stored, of a fully equipped character.



Figure 5.17: Fully equipped character inventory

Observe that the inventory screen should be planned to fit accordingly to the device that the game is developed for. In the case of mobile device games, we endorse a design that favours the touch input, having bigger images and touch buttons, and also to make use of touch gestures such as swapping, as it will definitely improve the interaction with the game. This observation fits every game screen that is susceptible of user interactions.

5.13 Items

Items are a very important trait of RPG games, greatly increasing the variety that a game has to offer. Items can range from wearable items that are usually called equipment or usable items that grants an instant or a temporary effect upon its user. They represent an important development effort as they are carefully designed one by one by dedicated designers. With this in mind, we created a item factory that allows creation of an infinity variety of items (with limited visuals) comfortably.

The idea of the factory is to have everything ready beforehand to be able to easily create the items on the go. To achieve that, we created a data blueprint for the items and for the equipment containing all necessary data of each different one. As of now, there are only equipment in the experimental game, so we will focus on them.

As stated before, an equipment is an item that the character can wear. While wearing it, the character usually receives bonus that increases their attack or defense. Since there is a lot of factors that can boost the character

overall strength, it is very beneficial to explore them all for equipment bonuses, as it will surely increase the variety of bonuses which makes the hunt for equipment much more thrilling. An example of an equipment named *Golden Skull Legs* data blueprint is expressed in the extract ahead.

```
Golden_Skull_Legs {
    attack: 0
    defense: 5
    autoAttack: 0
    maxHealth: 11
    critChance: 0
    critMult: 0
    autoSpeed: 0
    speed: 10
    rateDrop: 0
    rateExp: 0
    rateGold: 0.25
    chance: 0.08
    quality: 3
    slot: LEGS
}
```

From this extract, it is possible to realize some effects that the equipment can give to characters. They range from more direct effects that increases player's attributes, such as attack, defense and auto attack to more special effects such as increasing the rate of gold drop in the game. It is also present in the blueprint a parameter regarding the difficulty of obtaining the equipment, named *chance*. The lower this parameter, the harder it gets to obtain the equipment. Furthermore, the quality of the equipment is expressed from 0 to 3, with 3 being the highest quality possible. Finally, the category of the equipment, also known as slot, is defined within the *slot* parameter, and can range between the available types of equipment.

There are a total of 32 equipment blueprints in the experimental game. Each one with its own visual, category and effects. To surpass this limited variety of equipment, we utilized the strategy of assigning levels to the equipment. These levels affect their bonuses, and the blueprint works as the equipment in its initial level. The bigger the level is of an equipment, the bigger is the boost of its bonuses. The levels are randomly decided within a range that is related to the level of the enemy that dropped the equipment. The higher the equipment level, the harder it is to drop it. Since there is an infinite level progression in practice, there is also an infinite variety of equipment bonuses.

Figure 5.18 displays the *Golden Skull Legs* at level 25. It exposes how the level affected the effects of the item, specially comparing to the blueprint that equates to the effects of a level 1 *Golden Skull Legs*.

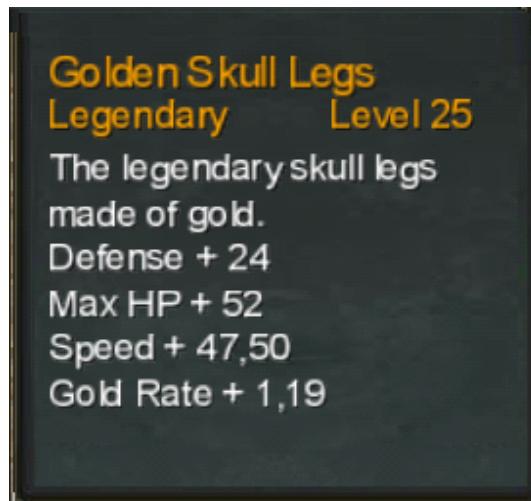


Figure 5.18: Golden Skull Legs effects at level 25

With this system of equipment level, we intend to introduce a new layer of equipment farming, as the player will be always wanting a better equipment that will always exist, and the difficulty progression will keep the player interested as the high level items will have a significantly increased value.

5.14 Enemies

Enemies are a vital part of most RPG games, seeing that they bring the challenge aspect to the players. They represent trials that the characters must overcome. Victory means reward, and defeat usually means punishment, this highlights how important an enemy is to the gameplay.

Similar to the items of a RPG game, enemies are usually a product of numerous design decisions, resulting in a huge development process. They possess unique visuals and traits, and the overall enemies quantity can easily exceed the hundreds. Since this is not an available possibility, we followed the item strategy in order to expand the variety feeling of the enemy encounters.

This results in enemies that are generated on the go, on the verge of each battle. Their visuals are randomly obtained from a pool of available enemy sprites, and their size gets bigger accordingly to their battle strength. Enemy's attributes are randomized within a range that is related to their levels. Each enemy generated has a random level that is based on the player's character current level. Their levels can be sometimes way bigger than the

player's character, or way smaller, improving the dynamic characteristics of the battle system. Based on their levels, the battle difficulty varies, as do the rewards. As the player's character evolves into higher levels, the enemies are more likely to be stronger, creating the need of obtaining stronger equipment throughout the journey. This makes the items even more valuable.

The last generated trait that an enemy has are their names. Even their names are generated, which improves the uniqueness feeling of each one of them. To generate their names, we used an algorithm based on Markov Chains. A Markov Chain is a way of generating a result based on the statistical probabilities in a set of sample data [56]. The name generator produces random names for the enemies based on a set of sample words used as an input.

Figure 5.19 shows an enemy that is result of the generation with all of its traits randomized, including name, attributes, and visuals. This tactic was very useful to increase the variety in the game and improve the player's experience without producing much development efforts, and with minor tweaks - such as avoiding certain generated names - can be used in major commercial games.

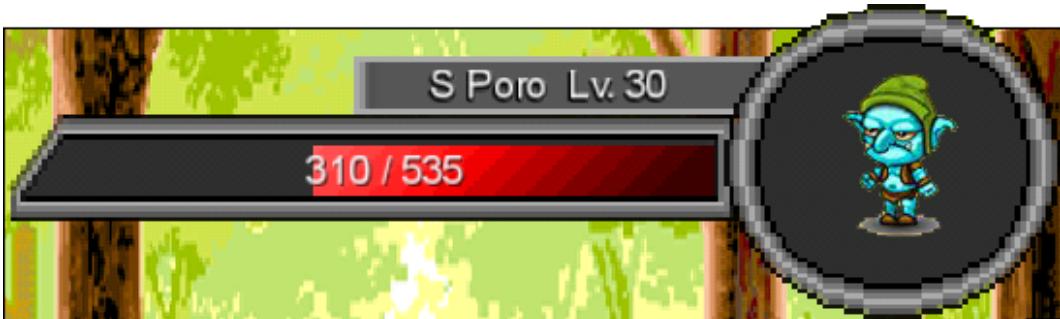


Figure 5.19: Enemy with random traits generated for battle

5.15 Battle

The battle system is responsible for matching the player's character with the generated enemies in a battle that will reward players if they emerge victorious. The battle happens in a separated screen in which the player will battle with the enemy encountered until one emerge as the winner.

The battle screen, shown in figure 5.20, is based on turn-based RPG titles such as Pokémon [28] while the gameplay of the battle is based on idle games [31] tap mechanics, in which players tap the screen to deal damage to their opponents. The faster they tap, the more damage per second they deal. Allied with that, both character's and enemy's attributes play a big role in the battle, as all the damage dealt are calculated through a formula that takes them into account. There is also an automated attack that deals damage without the



Figure 5.20: Battle screen showing the fight with a generated enemy

need of tapping the screen. This automated attack deals much less damage, but it is a plus to the overall damage the player can inflict. In addition, there is a chance to deal critical hits, which are hits that inflict more damage than normal. Both automated damage and critical hits are affected by the player's attributes. The battle ends when the health points of one of the contestants is depleted.

The described mechanic is very addictive as players strive to become stronger, increase the damage they deal by evolving or finding better gear, and to be able to defeat stronger foes. Games that follow that mechanic usually gather lots of players, such as *Clicker Heroes*, that in its PC version was able to top the Steam charts of active players at the same time in 2015 (figure 5.21).

If the player is the winner of the battle, rewards will be given to the player, including experience points, which are the means to get higher levels, and gold coins, which represents the in-game currency. There is also a chance that equipment are dropped as a reward. That chance relies on the *chance* modifier already discussed in the item section. This chance is affected also by the enemy's level, since the stronger the enemy is, the stronger the equipment reward is. This means that the level of the equipment dropped will fluctuate between a range that is around the enemy's level. The higher the item level, the harder it should be to get it, following a natural difficulty curve common in games.

CURRENT PLAYERS	PEAK TODAY	GAME
523,969	924,138	Dota 2
176,128	531,448	Counter-Strike: Global Offensive
91,301	131,965	Grand Theft Auto V
58,204	69,381	Team Fortress 2
29,092	41,242	Sid Meier's Civilization V
25,467	38,741	Garry's Mod
24,052	38,957	The Elder Scrolls V: Skyrim
23,515	38,295	Warframe
19,256	26,233	Clicker Heroes
18,500	57,823	Football Manager 2015

Figure 5.21: Steam charts of games with most current players showing Clicker Heroes

5.16 Ranking

In order to increase the player's motivation, it is important to encourage competition in games, specially in games with online features. Players tend to feel a higher sense of achievement when they compare their results with others. Nowadays, games that are mainly based on competitive characteristics are on the rise, with players becoming professionals and getting highly payed for their expertise on certain competitive games. This goes to show how competitiveness have a say in the player's motivation.

The main measurement of achievement in the experimental game are the character levels, as they represent the result of the whole adventure and all the conquests obtained throughout the world. Hence, the easiest way to introduce the competitive aspect to the game is to provide a ranking with the characters with highest levels, and to show the player's character position among the others. The ranking implemented for the game (figure 5.22) does exactly that, as it shows the top players in the game, allowing players to scroll through the highest level characters, while pointing out the current player position in the ranking.

Many other features can bring competitiveness to a RPG game. Duels between players, battle between guilds, events that rewards winning players, and many others. These elements should be considered with care, as they sure improves the player's overall experience if correctly done.

Ranking		
Position	Name	Level
1	Jotun	34
2	Brant	5
3	Sethim	5
4	Asger	3
5	Folke	2
6	Inkeri	1
7	Halvar	1
8	Borghild	1
9	Freja	1
10	Sven	1
11	Sigrid	1
12	Sigurd	1
13	Sigmund	1

Your Position: 1

Figure 5.22: Ranking with the game's top players

5.17 Market

Competitiveness is not the only aspect that enhances player's motivation. There is another aspect that for some players can work even better - cooperativeness. Players want to interact with other players not only in competitive manners, in fact, some of them might prefer to cooperate with others. Cooperativeness has the ability to transform a regular game into an amazing one, as the joy of playing with friends throughout the game's story in a cooperative mode overcomes minor existing flaws in the game.

In an online RPG, there are many elements that bring cooperative traits to the game. The most common are the formation of groups and guilds. They form a bond between players and enhance the social aspect of the game. Yet another way of increasing player's motivation, but each path chosen may attract players that would not be interested otherwise, so it is indeed important to analyze and decide what is worth from the perspective of the game design.

Allied with the goal of bringing a cooperative aspect to the game, we also intended to add an economy that is fueled by the infinite variations of equipment existent in the game. This strategy makes, once again, the equipment system much more valued, as it attaches monetary value to each equipment. In fact, one might say that an item system without some kind of economy system may fall short of its potential.

To create the economy of the game, we opted to implement a market system that allows players to buy and sell items with each other (figure 5.23). With the system, players can navigate through the items being sold at the *Buy* tab, as well as sell items at the prices they desire at the *Sell* tab. They can also retrieve any items that they are selling if wanted, through the *Listings* tab. This creates a dynamic economy that is based on the rarity of items. The market becomes another way of obtaining strong items, with the monetary value related to their usefulness. Note that some features such as filters and item search are strongly recommended to be implemented in a more robust market system, but the basic buying and selling operations is more than enough to start the economy, bringing life to the game's world.

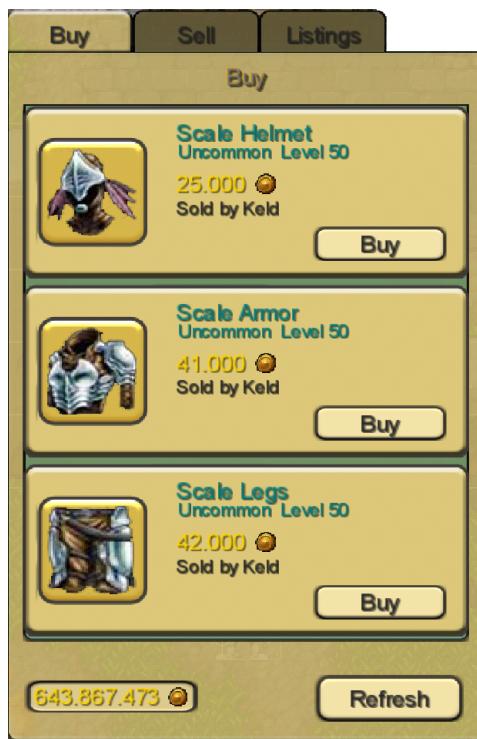


Figure 5.23: Market buy tab containing showing items being sold

5.18 Pervasive Map

To introduce the pervasive mechanics of the game, which revolves around the campus of PUC-Rio, a map visualization is presented as a representation of the pervasive world of the game, containing the current available pervasive locations guiding through the important pervasive tasks of the game.

In a RPG game, the map usually represents the game's world as it guides the player through the adventure. The same idea is applied to the pervasive map that is present in the game. The pervasive map of the game is the guide

for the game pervasive world, where players are encouraged to explore it by receiving rewards on world investigation.

The pervasive map can be moved, zoomed in and out as it allows players to interact with certain pervasive locations predefined. These locations are real world locations that are integrated in the game to serve as special spots that players can interact with. For each location, a brief informative description is provided serving as an educational guide of the campus. An example of a location is shown in the first image in figure 3.2.

For each pervasive location created in the pervasive map, numerous pervasive missions can be attached. These missions are equivalent to the usual RPG missions that players undergo to advance in the game's story and to become stronger. The difference is that the missions are completed through pervasive means, such as visiting certain locations or encountering secret hidden objects in the real world.

The mission system developed allows an easy creation of location based missions through direct parameters that provides a customization that makes the system very flexible, with features that are directly inspired by MMORPGs. The code ahead is an excerpt of a particular mission displaying some configuration parameters, serving as a guideline for defining similar systems.

```
-- is this mission repeatable?
missionRep: false
-- if it is repeatable, what is the interval (in mins) to repeat it
missionInterval: 0
-- mission normal rewards
missionExpReward: 450
missionGoldReward: 1800
-- mission drop rewards (ids separated by commas, see examples)
missionDropReward: {15002, 17002}
-- mission equipment level
missionEquipLevel: 12
-- mission prerequisites (missions that must be done before this one)
missionPreReq: {5}
```

The code shows some interesting characteristics of the system. A mission that is created within the system can be repeatable within a time interval, which means that players can redo the same mission receiving its rewards as long as the defined time interval is surpassed between each completion. This is very useful for pervasive tasks that need constant fulfillment, such as participatory sensing tasks or class attendance registration tasks. The use of the mission system for pervasive related mechanics as those just quoted is

better explained in section 3.2.

It is also possible to see some other parameters that are very important for the system to be properly able to create flexible missions that are interesting to players. To be able to adjust the mission reward is vital, and in the proposed system the rewards range from experience points and gold coins to equipment. Any combination of these may be a reward of any mission. In case of rewarding players with equipment, it is also important to be able to control what is the equipment level, a very important factor that will strongly affect the quality of the equipment, as stated before.

Finally, to be able to create chains of missions that are interconnected and form a more complex large mission, a trait that is usually present in MMORPGs, it is interesting to have a way of defining prerequisites of a mission, which consists of other missions that must be finished before enabling the current mission. These features are recommended for all systems alike, not only involving pervasive missions, but also normal missions as well.

5.19 Sensor Search

Sensors are the main vehicle of the game's integration with the real world environment. In this project, we used Beacon and SensorTag sensors. The SensorTag sensors provide temperature, humidity, accelerometer, gyroscope and other types of data, while the Beacon sensors only transmit a single signal that other Bluetooth-equipped devices that are in range can "see".

The sensors are responsible for the infrastructure behind the completion of every mission in the mission system described above. This is because it is necessary to connect with the mission attached sensor to finish it. For that, the player can scan the area for nearby sensors at any given moment, as shown in the second image of figure 3.2. The general flow of the mission system and the interaction with the sensors are detailed in section 3.2.

To be able to discover and connect with the IoT devices, the S2PA service is used, as explained in section A.3.2. The service is enabled and disabled accordingly to the players use of the sensor scan in-game feature. Once the service is on, it will discover and connect to nearby sensors, retrieving data that are collected by the IoT devices.

These data range from identifiers of the device to the data collected that varies depending on the device in question. The identifiers are very important as they serve as an unique representation of the device, helping the attachment of sensors to missions. The data is stored in an appropriated data structure, to be later sent to the game server and stored in the database.

The S2PA service, that is contained within the M-Hub API, is instantiated through native android specific code, which is apart of the libGDX framework game sub-project, as explained in section A.4. Native android code is within the android sub-project. Thus, an interface will be needed to access the sensor retrieved data. Fortunately enough, this can be done with few steps, with the whole interfacing scheme represented in figure 5.24.

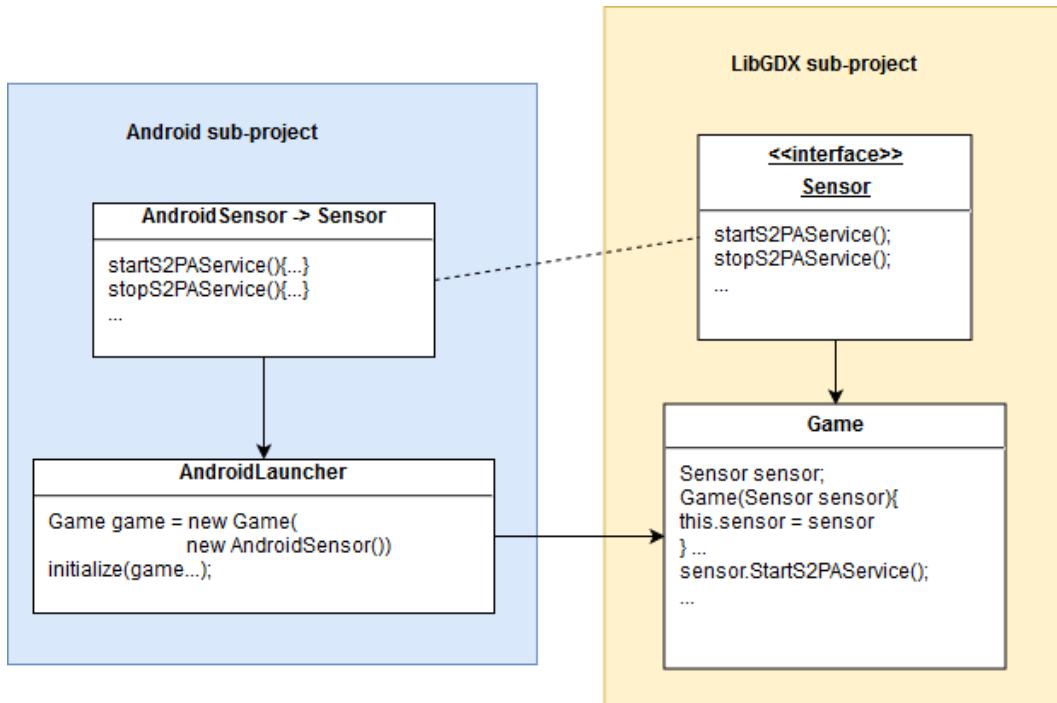


Figure 5.24: Native code interfacing scheme

Figure 5.24 shows the scheme that allows the integration of the S2PA service within the game libGDX sub-project. The strategy is detailed ahead for clarification.

1. Create an interface to act as a bridge for the necessary native code methods in the core libGDX sub-project;
2. Create a class in the android sub-project that implements the necessary native code methods as it implements the created interface;
3. When initializing the game class in the AndroidLauncher, make sure to pass the interface implementation class;
4. Use the available methods that are declared in the interface in the libGDX sub-project freely through the received implementation.

6

Evaluation

This chapter details our tests and results in public experiments where we deployed our experimental game for the public to test and assess their receptivity through an interview. We also analyze the game's latency and rendering performance and the effects of it on the overall player experience.

6.1

Experiments with Students

In this section, we describe the experiments we did with the public in the campus of our university, presenting the results of our survey for each experiment. The experiments aimed to evaluate the receptivity of the experimental game pervasive mechanics, focusing on the participatory sensing and the serious applications integration within the game mechanics.

A total of thirty subjects have participated in the experiments. Each subject was individually interviewed after the completion of the tasks, but some subjects did the tasks in groups of two or three. The majority of the subjects did not have previous knowledge about the sensors used in the experiment, so they were interacting with such devices for the first time.

Although the subjects were not using their own smartphone devices during the experiments, each subject was explained that for each sensing task performed, the 3G / LTE / 4G bandwidth of the device would be consumed. This way, we intended to elucidate the cost of such tasks before they were interviewed with questions related to their experience regarding the use of participatory sensing tasks in the game.

6.1.1

Experiment A

We provided a demo of the experimental game with a customized group of missions made specially for a public event where several high school students visited the university campus where they were exposed to departments, libraries, laboratories and other academic bodies of PUC-Rio. Our test subjects were undergraduate students from the university as well as local school students that were participating in the event.

The customized group of missions consisted of a circuit of missions composed of three parts and a special secret mission. The subjects were encouraged to play the game but also complete the pervasive missions available. Unfortunately, we did not have the structure to expand our area of tests further than the area that our stand was in, so the sensors used were placed somewhat near our stand. This limitation restricted the spatial guide tests, but we made sure to explain every aspects of our research that were not being directly tested to each subject of the survey.

The first part of the circuit would give players a tip to find the attached sensor that was in a special trash can adapted with a sensor that would detect when trash was thrown in the can. Once the sensor was found, players had to find out the mission code from a text that highlighted the importance of recycling was shown in-game when the mission sensor was accessed. The code was the combination of the first letters of the colors in a led panel in the adapted trash can that would ignite when a trash was thrown. In this mission, we explored the mission system in a educational way, by linking the mission with the notion of recycling.

The mission for the second part of the circuit would guide the player to an array of three physical boxes that were hidden at the stand. The code to complete the mission was written in the mission sensor that was hidden in one of the boxes. Players had to first solve a logical puzzle to find out in which box the sensor was hidden. Once they figured out, they were given the chance to open the box they thought the sensor was in. If they were correct, they would see the hidden code written in the sensor and use it to complete the mission, obtain the rewards and the access to the next mission of the circuit. In this mission, we tried to motivate the logical reasoning of players through a puzzle in the molds of logic subject, following the educational path of the first mission.

Finally, at the third mission of the circuit, participatory sensing was introduced, as the mission description detailed the notion behind it, and asked the players to share their bandwidth in order to retrieve the mission sensor collected data. Additionally, the mission required players to collect data that the sensor alone was not capable of obtaining. In the mission, they were told that it was necessary to retrieve the amount of men and the amount of women in the present moment in the stand, as an example of a data that was hard to obtain only by autonomous sensors, so they were tasked to count them and to provide their findings as an input of the mission, before they were able to finish it and receive the rewards. We made sure to acknowledge their contribution, and to give them a proper in-game reward.

The secret mission was an additional mission that could be completed without any prerequisites. We added this mission to intrigue players and to increase their interest in the game demonstration, since this mission not only provided the best in-game rewards, but also a secret real reward. On purpose, this mission difficulty was way higher than the others, with a two-part enigma involving logic and creativity.

For each subject, after they completed the main circuit of missions and had a minimum amount of experience with the game in general, we asked a few questions related to their reception of it. Those questions were particularly about their thoughts on the pervasive aspect of the game, as well as the combination with participatory sensing and serious applications. We obtained very interesting results that are described hereinafter.

When asked if the pervasive aspect of the game with the use of IoT devices had increased, decreased or did nothing to their experience in the experimental game, every subject of the survey responded that it positively increased their experience, showing the current receptiveness for the pervasive in the universe of games, and the potential of using IoT devices to expand pervasive practicability.

Over half of the subjects agreed that they felt more motivated to help in data collection when rewarded with in-game virtual recompenses, while 87.5% would feel the most motivated for such tasks when receiving rewards both virtually and physically (i.e. monetary reward). Only 12.5% said that they would feel motivated to comply in sensing tasks without receiving any kind of incentive. These results suggests that our collective sense of responsibility towards our society's improvement in the current state of mind of our population is lackluster, thus for participatory sensing effectiveness, external incentive still is very important. It also shows that there is a potential regarding virtual rewards as a valid incentive for participatory sensing tasks, and we could explore it along with other type of incentives.

In relation to the integration of serious applications with pervasive games, subjects were exposed to some applications such as lecture attendance registration and spatial guide of the campus. The reception was highly positive, as all of the subjects of the survey have concluded that it is interesting to ally real tasks that can be important or even necessary with in-game missions that rewards on realization of such tasks. Some of the subjects were so engaged in that matter that they would suggest various other applications that they thought would benefit of this integration, displaying the prospective that this approach has.

Lastly, as a result of collecting data through the mission system of the

game, as explained before, we were able to map varied data throughout the time period of the event. The data collected is just a small sample obtained by the experiment, but the technique can be used for large and more important data collection as well. The chart shown in figure 6.1 illustrates some data obtained by asking players to do the sensing task of counting the number of men and women at the stand by the time they reach the end of the last mission of the circuit.

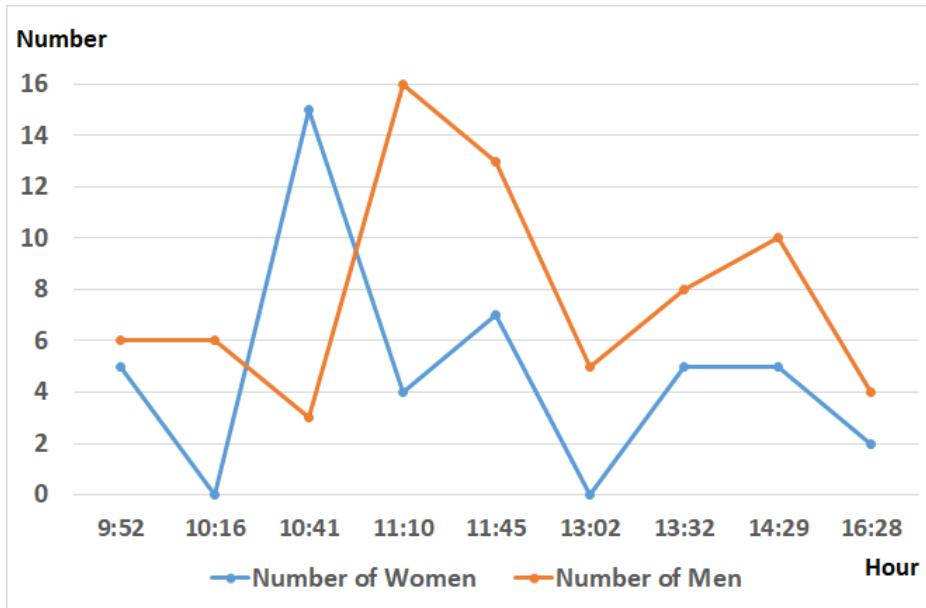


Figure 6.1: Chart representing number of men and women in the stand throughout the event

From the chart, we can see that the number of men in the stand is bigger than the number of women most of the time, although in some instants of the event the opposite happened. Additionally, there were data collected by the sensors that were also sent by the players, such as temperature and humidity. For a sample visualization, temperature time series can be seen in figure 6.2, while humidity time series can be seen in figure 6.3.

6.1.2 Experiment B

The second experiment we did also involved an array of missions in the game that the subjects did before answering a set of questions in a interview. Once more, it was done in the campus of PUC-Rio. The subjects were composed of students from engineering, design and computing courses. There were students from various semesters of their courses, ranging from students at first semesters of their courses to students that had already graduated. Of the subjects interviewed, 71.4% had an age between 18 and 24 years old, while

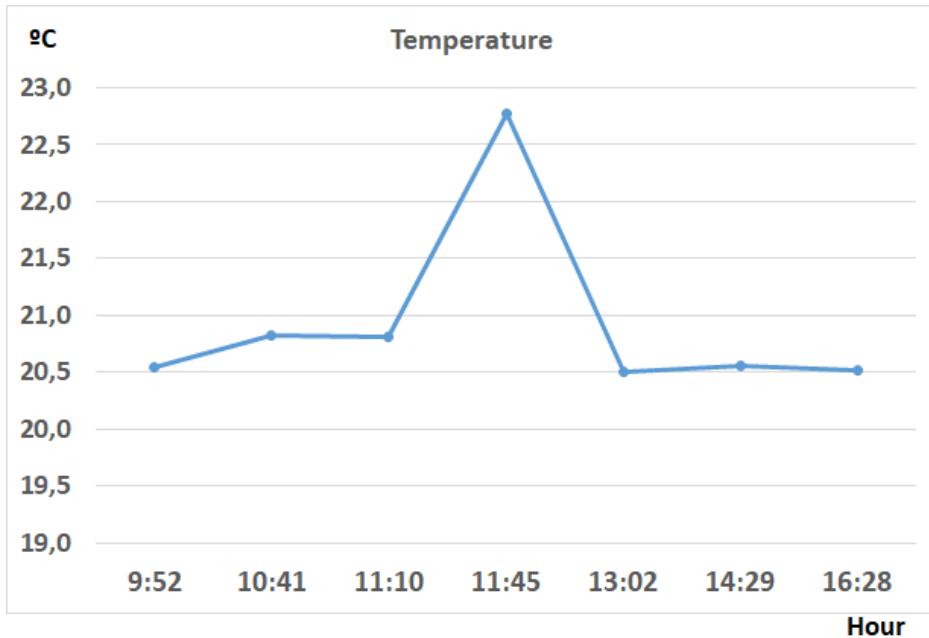


Figure 6.2: Temperature time series throughout the event in the stand

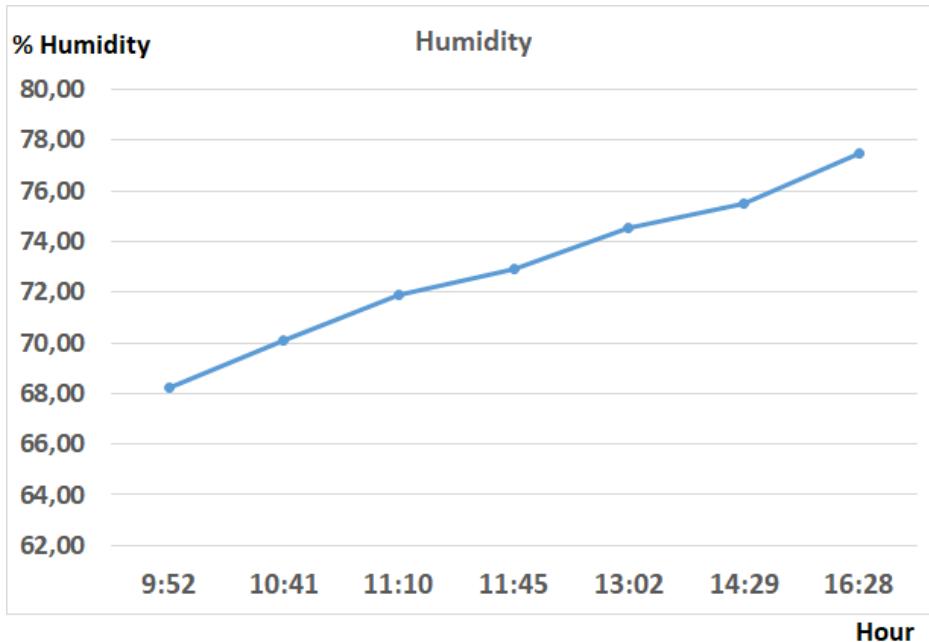


Figure 6.3: Humidity time series throughout the event in the stand

23.8% were between 25 and 35 years old. The 4.8% remaining were between the interval of 36 and 50 years old.

For this experiment, there were three missions that the subjects completed in sequence, as each part was prerequisite to the next one. The first mission from the first experiment, where the sensor was attached to a modified garbage bin that had a led panel that would light up when trash was thrown in the bin, was recycled for this experiment, but this time the trash

bin was placed in another location for subjects to find, in one of the laboratories of the campus. In this mission, the code needed to obtain the in-game reward and unlock the next mission was the combination of the first letters of each color of the led panel. By describing the importance of recycling in the mission, we intended to show an educational usage of the system, while providing an interesting interaction with physical world objects that can increase the experience of a game.

The second mission consisted of finding a secret code that was a word inside a specific book. The goal of the mission was to display a serious application integration, where the central library of the campus was the targeted facility. Each subject had to discover what was the specific book through a riddle before proceeding to the central library for retrieving a copy that was borrowed for the experiment. They were given a code that served as a tip to find the secret word in the book. With the secret word obtained, the mission could be finished and the rewards would be given. This mission presented the central library for students that were not aware of the library services that were available to them. By guiding the subjects to the library location through the mission, the spatial guide application was also tested. While in the library, students that did not have registration in the library system seized the opportunity to create their registrations, as they were guided through the library services.

The final mission involved the integration of participatory sensing into the mission system. This time, we aimed to find a sensing task that the subjects would relate and see the benefits of its completion. One of the problems that is common in the campus is to find an available bathroom that is in good condition, with students often walking more than the necessary and wasting time to find a suitable one. Therefore, the notion of having a quality control of bathrooms in the campus by providing a map that displays the working bathrooms and their conditions, would be something the subjects would relate to, and also be an application that would benefit of participatory sensing. To express this application in the last mission, we placed sensors in two bathrooms for the experiment (one masculine and one feminine bathroom), and the mission completion would solely depend on an inspection of the targeted bathrooms. Each subject were tasked to rate the bathroom at the time of the experiment in four different aspects that were related to the cleanliness, the scent, the supplies and the availability of toilets. They were told the meaning behind the task and how they would be contributing in the campus after they were given in-game rewards, finishing the last mission of the second experiment.

After the completion of the missions and the familiarization of the game

and the concepts that were presented to them, the subjects were prompted to answer few questions related to their reception of the experiment, regarding the pervasive aspect of it and the integration of the participatory sensing and also the serious applications into the gameplay. The results are described hereinafter.

90.5% of the subjects stated that they had a good overall experience with the experimental game, while 9.5% felt that the experience was average or just above average. Furthermore, something similar to the first experiment happened, as all the subjects answered that the addition of the pervasive mechanics had positively increased their experience with the game. With few months apart between the two experiments, this shows that the receptiveness of pervasive mechanics did not fade out just yet, and that the spot for pervasive games has been cemented in a way that it will possibly last a while.

Almost every subject (95.2%) stated that a virtual or a real incentive would make them more motivated to realize participatory sensing tasks such as the one that they did in the experiment. When asked if only virtual rewards would be enough to motivate them to comply with the tasks, 61.9% of the subjects said yes. Once again, this displays the potential of exploring the high motivation of pervasive game players to cover a large amount of sensing tasks. The increase in the positive outcome of the second experience in regards of the integration of participatory sensing tasks into the pervasive mechanics is another indicator of the ongoing receptiveness of pervasive mechanics in games.

In regards to the integration of serious applications, the same positive response was observed. 100% of the subjects have shown interest in the union of real and necessary tasks into the game mechanics. This highlights the potential of such approach that can be explored for numerous applications and for numerous other facilities e.g. museums, thematic parks and public libraries.

The data collected throughout the experiment related to the bathrooms used in the last mission is exposed in the charts displayed in figures 6.4 and 6.5. The bathrooms chosen for the experiment are usually in good condition, which is reflected in the average score shown in the charts. We can also see that the female bathroom was in a better condition throughout the experiment.

6.2 Performance Tests

The tests described in this section are related to the performance of the game, and will include tests for the latency of the communication and the game rendering performance as well.

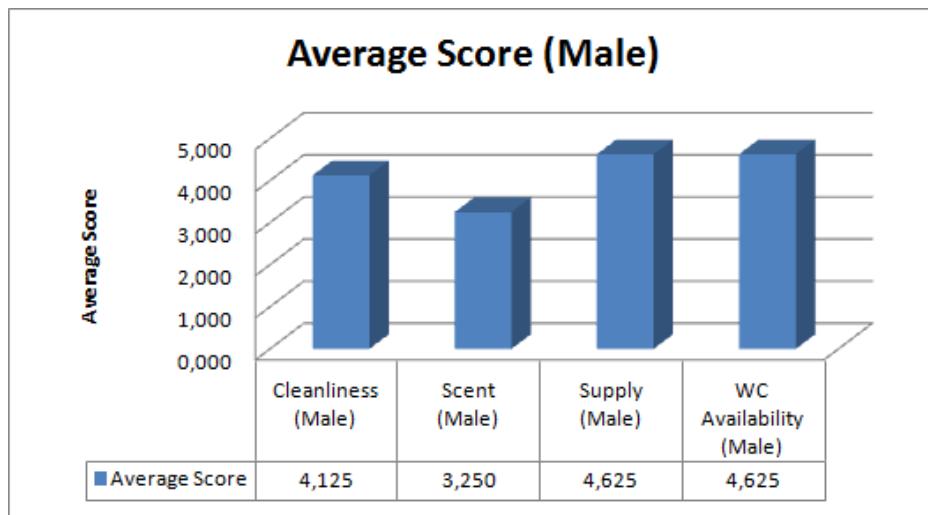


Figure 6.4: Average score of the male bathroom used in the experiment

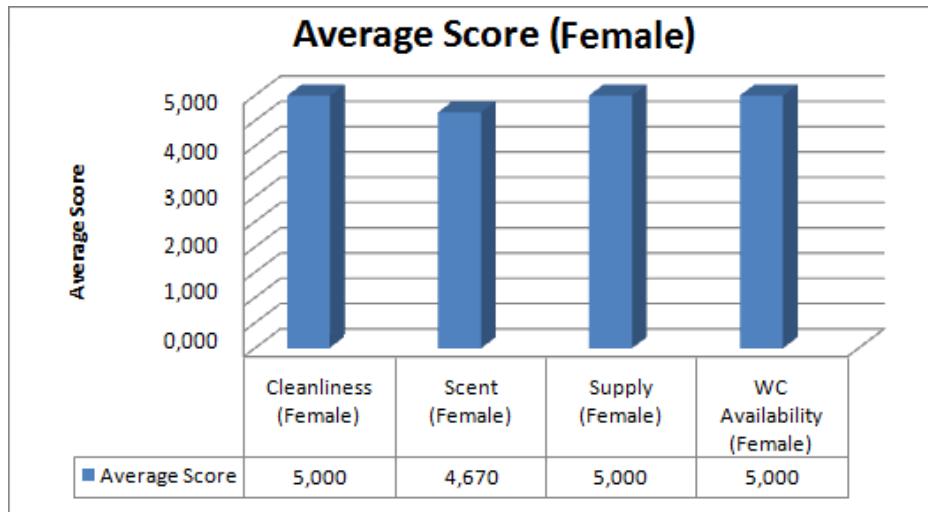


Figure 6.5: Average score of the female bathroom used in the experiment

Tests were done using *Motorola Moto G* first generation smartphone, with the following specifications:

- *Model* - Motorola XT1032;
- *OS* - Android 4.4.4;
- *CPU* - Quad-core 1.2 GHz Cortex-A7;
- *GPU* - Adreno 305 450MHz;
- *RAM* - 1 GB;
- *Resolution* - 1280x720;
- *Flash Size* - 8 GB.

The game server was hosted in a machine with the following specifications:

- *OS* - Microsoft Windows 8 (build 9200), 64-bit;
- *CPU* - Intel(R) Core(TM) i7-3840QM CPU @ 2.80GHz;
- *GPU* - NVIDIA GeForce GTX 680M;
- *RAM* - 8 GB;
- *HDD* - 931.2 GB;

6.2.1

Latency Tests

Although it is an online game, the experimental game is not in real-time. The online features present in the game are synchronous and do not interfere directly in the core gameplay as players use them in calm and non-critical situations. This means that client-server communications latency would affect less the player experience, but we still aimed to reduce latency time to enable a smoother experience, including sensor discovering and connecting latency through the S2PA service, in which we defined a scan interval that allowed a fast and smooth interaction with nearby sensors.

The table 6.1 shows latency test results for different client-server operations in the experimental game. Numerous latency tests were made to obtain an average value to each operation. Tests were made with different characters with distinct progress in the game, resulting in a varied amount of data for each operation, in order to better represent the latency players experience throughout the game. These operations and their average latency time are exposed in the client-server latency test table hereinafter.

Operation	Average Time
Server Registered Sensors Retrieving	206ms
Player Completed Missions Retrieving	146ms
Complete Mission Operation	258ms
Ranking Retrieving	186ms
Login Operation	240ms
Market Retrieving	190ms

Table 6.1: Client-server latency tests (ms for milliseconds)

Results shown in table 6.1 takes into account all operations that are made in server side for each request, with the goal to better elucidate the latency time for client-server functionality of the game. This means that any needed working time for algorithms in server side are also included in the presented latency times. It is also important to highlight tests connectivity circumstances, where the client device used 3G connectivity, and the server connection floats

around 20 megabits per second of download speed and 5 megabits per second of upload speed.

As we can see in the values shown in table 6.1, all average times for the exposed operations were lower than 300 milliseconds, which is pretty acceptable for operations that are not in real-time, which is the case for all client-server communication for the experimental game.

6.2.2 Rendering Performance

Usually, game rendering is active and in real-time, resulting in a constant refresh of the rendered screen. This refresh rate is normally called FPS (Frame Per Second), representing the amount of frames that are being rendered per second, and is the most common used measurement of a game's rendering performance. Developers nowadays strives for reaching the mark of 60 FPS in their games, as players are getting used to this threshold.

The game's experience can be severely hurt by low FPS rendering and FPS drops that occur in certain situations of the game, so we aimed to achieve a near constant FPS throughout all game screens, revolving around the 60 FPS mark. To be able to do so, the FPS was constantly monitored throughout the game development as a way to avoid the introduction of any anomalies that would affect the game's rendering performance. Table 6.2 shows the test results of the FPS for different game situations.

Screen	Average FPS
Initial Menu	61 FPS
World Navigation	61 FPS
Battle	60 FPS
Inventory	61 FPS
Ranking	58 FPS
Market	60 FPS
Pervasive Map	61 FPS
Sensor Search	61 FPS

Table 6.2: Game screens FPS rendering tests

As seen in the table 6.2, all average FPS results are very close to the 60 FPS mark. This is partially due to the fixed time between frames that sets the rendering to 60 FPS. This means that even when the rendering can be done at higher rates, the rate will still be close to 60 FPS. Naturally, if the rendering cannot be done at the desired rate, the FPS sure will drop.

After several optimization such as clipping unnecessary entities from rendering, as well as the carefully management of resource initialization, the

goal of achieving the 60 FPS mark with very low variation was achieved. As a consequence, the game runs smoothly without any bumps in rendering performance and without affecting the experience for the player. We conclude that libGDX is a nice solution for developing Android games, with a good variety of development resources and results that are proven to be efficient.

7

Conclusions

Throughout this work we developed a pervasive IoT game that demonstrated the integration of several useful technologies, tools and a distributed middleware to develop a pervasive game using some types of IoT devices with sensors, beacons and wireless WPAN connectivity. We also explored and discussed the possibilities of incorporating participatory sensing tasks and serious applications in pervasive games, as we carried out surveys that assessed the public response of the proposed pervasive game.

The public receptivity showed us the prospects of uniting serious application and participatory sensing tasks with pervasive games, as the results exposed in chapter 6 indicates that player's motivation can be explored both for participatory sensing and for serious applications. But care should be in mind in the case of sensing tasks that requires players to use their reasoning to be the agent for the data collection, where the data precision is of high importance, as we advocate that the simpler the task, the more precise the data collection is, and the less inaccurate data is obtained. In summary, an equilibrium must be found where the labor of sensing tasks does not surpass the motivation that the game provides.

The latency test results showed that using the ContextNet middleware to aid in client-server communication for games with similar online features is viable, and was very positive as it provided a robust architecture that helped the development of the online elements of the game, e.g. market system, and decreased the game development effort. With that in mind, we recommend the use of the ContextNet middleware for the development of games with online capabilities such as the game proposed in this project.

In our controlled environment, the proposed game has shown a lot of promise, but this potential is not restrained to only our university campus, as the game universe can be applied to any environment that would benefit of such applications. We also concluded that the spatial guide in combination with the missions that connects real locations within the game has shown a valid way of providing educational content to players, since the mission system is able to describe each location in it. Moreover, this approach provided a very functional spatial guide that can be extended to any environment that

demands for such application.

In the participatory sensing tasks of the experiments, the human senses of vision and smell had a huge importance. For instance, the sensors in the bathrooms could only guarantee the proximity of the subjects, but were not capable of evaluating the current state of each bathroom. More capable sensors can be used as an alternative, such as an odor sensor, but the costs of those devices may be out of budget. Also, the human senses vary from person to person, e.g. smell and palate senses, thus, using human senses should give a more representative sensing data in those cases.

Other than the survey results we obtained, we gained unexpected but important knowledge regarding the structure of a pervasive IoT application. As stated before, the demo special pervasive missions were limited within an area from our stand. A big part of this limitation was due to the fact that we needed to make sure that our research sensors were safe from impacts and theft, so we defined a limited area for the tests in order to guarantee greater control of the environment during the observations. This goes to show that unexpected side effects of pervasive applications can always surprise us, and we must try to anticipate them the best way we can. In a long term, there are other factors that are important to consider before preparing a location with IoT devices, such as avoiding places that should not be disturbed by player visits, avert damaging fauna and flora of locations, and prepare it for climate variations.

Although the need of moving while playing the experimental pervasive game proposed is shortened by the fact that the interactions with the real world through the IoT devices does not require players to pay attention to the smartphone constantly, as they can calmly read missions and guide themselves through the map without the need to be agile nor opportunist, we still defend that is very important to be careful with player safety when it comes to pervasive games, as we learned how devastating the consequences can be when pervasive games are played indiscriminately, with studies showing how the pervasive game Pokémon GO has caused numerous accidents, including fatal ones [32, 33]. Other fact that reduces the risk of accidents is that the game's pervasive map is within a controlled environment, but bear in mind that even in a controlled environment any type of pervasive interaction with it must be implemented carefully, avoiding unnecessary risks. With that in mind, we support that the use of a very strict speed limit should always be considered as an option to increase the safety of pervasive games.

In conclusion, the integration proposed with the technology used fulfilled our expectations, and the effort into using IoT devices as an extent of the

pervasive interactions of our experimental game has produced promising results, as it not only improved the precision of real locations mapping within the game, but it also expanded the possibilities of integration with other applications such as participatory sensing and serious applications, in which we observed a lot of potential that is shown by our surveys.

7.1 Future Works

This section presents some future works that can act as a follow-up and enrich this research, as several questions emerged from this work.

Firstly, as mentioned before, there were several unexpected side-effects of using IoT devices alongside with pervasive mechanics in the game, such as the possibility of external damages to the devices placed in the campus, as well as the chance of theft cases. There are many other factors that should be well thought before extending the game's world to a real location, which includes also the impact of the locations chosen in the real world, as they can negatively affect an area that may be of huge importance, such as hospital areas. For future works, it is very interesting to study ways of dealing with these factors, as well as research for others that can affect the devices and their surroundings.

Following the same logic, but now focusing on the player's safety, an investigation of forms to increase the safety in pervasive games is of extreme importance, as when pervasive games are played indiscriminately accidents are bound to happen. A careful examination should be made in order to reduce those chances. The more risking factors identified the less likely to have serious accidents involving players of pervasive games.

Those studies should follow the excellent research presented by the authors of [3, 4, 10].

The access to each sensor used in the experiment happened once at a time, as only one smartphone device was used for each experiment. Thus, there were no concurrent access with the sensors. For future experiments, it is interesting to create an environment that allows the concurrent access test and evaluation. In addition, it is interesting to perform an experiment that evaluates the mobility aspect of the system, as well as the scalability, which is a feature of the ContextNet middleware.

More experiments can also be done regarding the pervasive mechanics that includes participatory sensing tasks and serious applications, focusing on player experience analysis. For those experiments, others real world scenery should be tested, extending the university campus world, in order to find more

applicable scenery, aiming to extend the reach of such applications to cover a larger amount of player's surroundings. Another topic that can be explored is the use of augmented reality which is very common in pervasive games. The augmented reality feature was not part of this research as it is very prevalent already in pervasive games, but new ways of using this feature as a tool for helping sensing tasks data collection in a way that also adds to the game experience can also be investigated.

As of now, the project only communicates with sensor devices, through Bluetooth technology. Everything necessary to enable this communication is achieved through ContextNet middleware and its MobileHub extension. There is another very promising extension for the middleware that is soon to be available called M-ACT, which aims to enable communication with actuators in a very efficient manner, by providing a language pattern that helps the communication. The integration of the M-ACT extension with the other components in this project is very beneficial, and can be done the same way MobileHub was integrated (through the native code interface, explained in section 5.19). Further studies on ways of using actuators to increase player experience can also be done, as it opens a whole new world of gameplay possibilities.

For the game in question, there are numerous upgrades that can be done to better the game experience, following the manners of the RPG and MMORPG genre. Some of them are hereby listed for similar future works:

- Implementation of an architecture that allows real-time online physical interaction between the characters in the game;
- Design a more detailed character creation, which allows players to choose the look of the character;
- Provide different types of battle classes that a character can choose, so that the player can follow the path of their preference;
- Implement special battle abilities that characters can have and use on battle;
- Dynamically change the look of the character according to your equipment;
- Allow battles between players;
- Provide systems that increases the cooperation between characters, such as groups and clans.

A study on other types of games that can also be integrated with IoT devices and pervasive tasks that enables the interaction with the real world

should also be headed. There are probably other genres that fit very well within this universe. We suggest that genres that have the interaction with the scenery as a major gameplay factor, such as investigation games, may have a natural and smooth integration with the real world as it becomes the scenery of the game, where all the in-game investigation can be related to the real world scenery. Games that are heavily based in its story and dialogs may also be included in this pack, since those games are strongly connected to the universe in which the game world is present, which leads to an interactive experience with the game's world that can be transported to a real world scenery.

7.2 Recommendations

Here we present some recommendations based on the experience of developing the experimental game, as well as the experience with the tests that were done with the public. They relate to the use of IoT devices as well as the integration of participatory sensing tasks and serious applications. The recommendations are the following:

- Align development goals with entities involved in pervasive interactions before beginning development;
- Avoid populating with sensors dangerous areas that may offer risks to the player;
- Avoid populating with sensors areas that the movement of players can disrupt vital services such as health;
- Protect the physical integrity of the IoT devices avoiding damages caused by climatic actions, shocks, among others;
- Protect the IoT devices against theft;
- Maintain a systematic check for exchange of defective batteries or devices;
- Focus on tolerance of server connectivity issues, that is, if there is no connection at the time of data collection, store it until the connection is reestablished, avoiding the loss of data specially on isolated areas;
- Inform the player about the data collection process in which he will be part of by lending his bandwidth or by being the main agent of the collection, before sending any data collected to the server;
- Avoid asking complex and time-consuming sensory tasks to players in order to decrease the amount of unreliable data obtained;

- Reward the player fairly according to the sensory tasks performed, in order to not affect his experience with the game, and motivate him to perform other tasks of the same type;
- Check the integrity of the data collected in sensing tasks by cross referencing data obtained in similar moments in time, possibly rewarding the ones that are closer to the average and punishing the ones that are farther;
- Check the feasibility of integrating serious applications with the game (e.g. communication with third party database) before proposing it;
- Study the benefits of the integration of a serious application to make sure that it is worth, taking a special look at side effects that may occur;
- Ensure the security of the most sensitive applications, avoiding the possibility of abuse of system failures. The security of the system may have a much greater weight in the context of serious applications.

Bibliography

- [1] TALAVERA, L. E.; ENDLER, M.; VASCONCELOS, I.; VASCONCELOS, R.; CUNHA, M. ; E SILVA, F. J. D. S. The mobile hub concept: Enabling applications for the internet of mobile things. In: PERVERSIVE COMPUTING AND COMMUNICATION WORKSHOPS (PERCOM WORKSHOPS), 2015 IEEE INTERNATIONAL CONFERENCE ON, p. 123–128. IEEE, 2015.
- [2] ENDLER, M.; BAPTISTA, G.; SILVA, L.; VASCONCELOS, R.; MALCHER, M.; PANTOJA, V.; PINHEIRO, V. ; VITERBO, J. Contextnet: context reasoning and sharing middleware for large-scale pervasive collaboration and social networking. In: PROCEEDINGS OF THE WORKSHOP ON POSTERS AND DEMOS TRACK, p. 2. ACM, 2011.
- [3] VALENTE, L.; FEIJÓ, B. ; DO PRADO LEITE, J. C. S. Mapping quality requirements for pervasive mobile games. Requirements Engineering, 22(1):137–165, 2017.
- [4] VALENTE, L.; FEIJÓ, B.; DO PRADO LEITE, J. C. S. ; CLUA, E. A method to assess pervasive qualities in mobile games. Personal and Ubiquitous Computing, p. 1–24, 2017.
- [5] ESTRIN, D.; CHANDY, K. M.; YOUNG, R. M.; SMARR, L.; ODLYZKO, A.; CLARK, D.; REDING, V.; ISHIDA, T.; SHARMA, S.; CERF, V. G. ; OTHERS. Participatory sensing: applications and architecture [internet predictions]. IEEE Internet Computing, 14(1):12–42, 2010.
- [6] DE CRISTOFARO, E.; SORIENTE, C. Participatory privacy: Enabling privacy in participatory sensing. IEEE network, 27(1):32–36, 2013.
- [7] TALASILA, M.; CURTMOLA, R. ; BORCEA, C. Crowdensing in the wild with aliens and micro-payments. IEEE Pervasive Computing, 15(1):68–77, 2016.
- [8] XIAO, Y.; SIMOENS, P.; PILLAI, P.; HA, K. ; SATYANARAYANAN, M. Lowering the barriers to large-scale mobile crowdsensing. In: PROCEEDINGS OF THE 14TH WORKSHOP ON MOBILE COMPUTING SYSTEMS AND APPLICATIONS, p. 9. ACM, 2013.

- [9] EBLING, M. R. Pervasive computing and the internet of things. *IEEE Pervasive Computing*, 15(1):2–4, 2016.
- [10] VALENTE, L.; CLUA, E.; FEIJÓ, B. ; DO PRADO LEITE, J. C. S. The law of unanticipated consequences in pervasive games. 2016.
- [11] PAPANGELIS, K.; METZGER, M.; SHENG, Y.; LIANG, H.-N.; CHAMBERLAIN, A. ; KHAN, V.-J. "get off my lawn!": Starting to understand territoriality in location based mobile games. 2017.
- [12] GREGORY, M. Ingress: A game, lifestyle and social network in one. When In Manila. 2014-07-22.
- [13] NIANTIC. Ingress (digital game). Google, 2012.
- [14] WINEGARNER, B. Forget pokémon go, there's another augmented reality game that's way better. Quartz. 2016-07-15.
- [15] SMITH, J. A million people around you are playing an alternate reality game you can't see. Mic. 2015-05-28.
- [16] WEBER, R. Niantic's ingress has over 14m downloads. gamesindustry.biz. 2016-01-28.
- [17] TAKAHASHI, D. Pokémon go is ‘the most successful mobile launch in history,’ superdata says. VentureBeat . 2016-08-24.
- [18] LANDI, M. Pokemon go app named as the most popular game of 2016 on google play. Mirror. 2016-12-01.
- [19] WINDELS, J. Pokémon go figure – a data analysis of the most popular game of all time. Wandera. 2017-02-28.
- [20] WEINBERGER, M. 14 things you didn't know about pokémon go and how it was made. Business Insider. 2016-07-15.
- [21] WEINBERGER, M. The fad may be over, but pokémon go still has 65 million monthly active players. Business Insider. 2017-04-04.
- [22] RUSSEL, J. Pokémon go has now crossed \$1 billion in revenue. TechCrunch. 2017-02-02.
- [23] ETHERINGTON, D. Niantic's follow-up to pokémon go will be a harry potter ar game launching in 2018. TechCrunch. 2017-11-08.

- [24] WEBSTER, A. The walking dead is getting a pokémon go-style ar game. *The Verge*. 2017-08-29.
- [25] LAC. Contextnet middleware wiki. Laboratory for Advanced Collaboration. 2017-11-28.
- [26] LAC. Contextnet middleware wiki: Mobile hub. Laboratory for Advanced Collaboration. 2018-04-12.
- [27] MACDOWELL, A.; ENDLER, M.. Internet of things based multiplayer pervasive games: An architectural analysis. In: INTERNET OF THINGS. USER-CENTRIC IOT, p. 125–138. Springer, 2015.
- [28] FREAK, G. **Pokémon**. Japan: Nintendo, 1996.
- [29] SAMPAIO, P. 2dmapbuilder documentation. 2DMapBuilder Wiki. 2017-06-17.
- [30] DICKEY, M. D. Game design and learning: A conjectural analysis of how massively multiple online role-playing games (mmorpgs) foster intrinsic motivation. *Educational Technology Research and Development*, 55(3):253–273, 2007.
- [31] NESTERIUK, F. T. M. S. Action rpg e idle games: possíveis combinações assimétricas de mecânicas de jogo divergentes.
- [32] FACCIO, M.; MCCONNELL, J. Death by pokémon go. 2017.
- [33] AYERS, J. W.; LEAS, E. C.; DREDZE, M.; ALLEM, J.-P.; GRABOWSKI, J. G. ; HILL, L. **Pokémon go—a new distraction for drivers and pedestrians**. *JAMA internal medicine*, 176(12):1865–1866, 2016.
- [34] MULLENIX, D.; FULTON, J.; BROOKE, A. ; WINSTEAD, A. Explanation of gps/gnss drift. 2010.
- [35] REILLY, L. Universal anuncia game mobile de jurassic world parecido com pokémon go. IGN. 2018-03-09.
- [36] UEYAMA, Y.; TAMAI, M.; ARAKAWA, Y. ; YASUMOTO, K. Gamification-based incentive mechanism for participatory sensing. In: PERVASIVE COMPUTING AND COMMUNICATIONS WORKSHOPS (PERCOM WORKSHOPS), 2014 IEEE INTERNATIONAL CONFERENCE ON, p. 98–103. IEEE, 2014.

- [37] ARAKAWA, Y.; MATSUDA, Y. Gamification mechanism for enhancing a participatory urban sensing: Survey and practical results. *Journal of Information Processing*, 24(1):31–38, 2016.
- [38] GROH, F. Gamification: State of the art definition and utilization. Institute of Media Informatics Ulm University, 39:31, 2012.
- [39] ZECHNER, M. Introduction to libgdx. LibGDX Documentation. Acesso em 2018.
- [40] MDN. Noções de tile e tilemaps. MDN web docs. 2015.
- [41] SAMPAIO, P. M2d format. 2DMapBuilder Wiki. 2017-06-22.
- [42] ZECHNER, M. Gallery. LibGDX Gallery. Acesso em 2018.
- [43] SLANT COMMUNITY. What are the best 2d game engines for android? Slant 2D Android Engine Recommendations. 2018.
- [44] SLANT COMMUNITY. What are the best ides for android development? Slant Android IDE Recommendations. 2018.
- [45] LAC. Contextnet middleware download. Laboratory for Advanced Collaboration. 2018-06-04.
- [46] GRADLE INC. Gradle user manual. Gradle Docs. Acesso em 2018.
- [47] ZECHNER, M. Dependency management with gradle. LibGDX Wiki. Acesso em 2018.
- [48] PHPMYADMIN CONTRIBUTORS. Bringing mysql to the web. phpMyAdmin Website. 2018.
- [49] COBURN, M. Top 5 open source tools for mysql administrators. InfoWorld. 2017.
- [50] OPITZ, D. Xampp - replacing mariadb with mysql. GitHub. Acesso em 2018.
- [51] LAC. Contextnet middleware tutorial. Laboratory for Advanced Collaboration. 2017-07-21.
- [52] ORACLE. Java jdbc api. Java Documentation. Acesso em 2018.
- [53] ZECHNER, M. Class i18nbundle. LibGDX API. Acesso em 2018.
- [54] ZECHNER, M. Sound effects. LibGDX Wiki. Acesso em 2018.

- [55] ZECHNER, M. **Orthographic camera**. LibGDX Wiki. Acesso em 2018.
- [56] GAMES, S. **Markov name generator**. Silicon Commander Games. Acesso em 2018.

A

Setup Guide

This chapter aims to guide through the integration of all components used to develop the proposed IoMT pervasive game, therefore it guides for the preparation of the development environment.

A.1

Android Studio Setup

Android Studio IDE was chosen for the development of the IoMT pervasive game, so all integration that is described in this chapter will consider that. Thus, it is recommended that the same IDE is used for any project that wishes to follow the integration described ahead.

When it comes to setting Android Studio up, the only concern is to make sure to include the Android SDK on the installation, as it is necessary to the development of the application. In case you already have the SDK, you do not need to install it again, but you will have to link the SDK location in your computer on Android Studio configuration, as shown in figure A.1. You may use any Java Development Kit version 8 or newer, but you can use the embedded JDK version 8.

We will not be creating our Android Studio project through the IDE, instead we will prepare the project beforehand in order to integrate the framework LibGDX. This is due to the existence of a project generator for the framework that reduces the effort into creating a LibGDX project.

A.2

LibGDX Project

To aid the development of projects that use LibGDX framework, LibGDX project generator (figure A.2) creates a template project with all the necessary components to enable the use of the framework. In the generator, it is possible to pick the name of the project, package, main class, as well as choose the project destination path, but you must ensure that the Android SDK location is correct.

Once the project is generated, the next step is to import it into Android Studio.

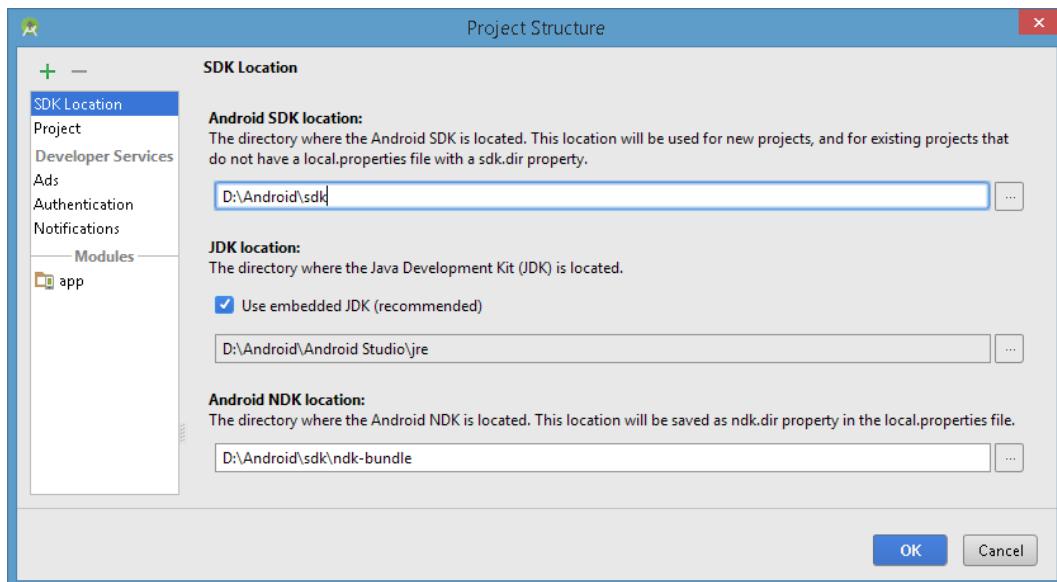


Figure A.1: Android Studio SDK Location configuration



Figure A.2: LibGDX Project Generator

A.3 ContextNet Setup

To better structure ContextNet's Setup guide, we divided into two different sub-guides: the first one details the setup for the SDDL Core (that runs the server side), and the second one details the setup for the Mobile Hub extension of the ContextNet middleware.

A.3.1

SDDL Core Setup

The ContextNet middleware is distributed as a set of software modules in the form of JAR files. In order to develop applications based on ContextNet services, you will need to include these libraries in your application build path [45]. In the case of our application, which is a LibGDX project, we must use Gradle to include these libraries, which will be explained further in the Gradle Setup section.

To provide real-time communication services, the ContextNet's middleware communication layer, Scalable Data Distribution Layer (SDDL), uses the Data Distribution Service (DDS) protocol in its core network and the MR-UDP protocol in the edges [45]. It is necessary to utilize some DDS product. For the latest ContextNet, version 2.7, OpenSplice 6.7 is the recommended option for an open source implementation of DDS. A detailed tutorial on how to install OpenSplice can be found at the ContextNet's tutorial page [45]. Additionally, it also requires Sun/Oracle Java 1.6 runtime or newer.

To execute a ContextNet application, one needs to first create the gateway, which will instantiate the core infrastructure if none exists. You can create a gateway by running the ContextNet middleware jar file. The gateway receives three parameters: a public IP address, a given port number, and a DDS vendor implementation to be used [45]. Figure A.3 serves an example of a command line that locally deploys the gateway through loopback IP "127.0.0.1". A helpful tip is to create a batch file with the command line to save time when deploying the gateway.

Once the gateway is running, ContextNet's SDDL will be fully usable, and will provide the means to distribute messages between the game's client and the game's server in a very reliable manner.



```
C:\ Administrador: Prompt de Comando
Microsoft Windows [versão 6.1.7601]
Copyright © 2009 Microsoft Corporation. Todos os direitos reservados.

C:\Users\MICRO>java -version
java version "1.8.0_121"
Java(TM) SE Runtime Environment (build 1.8.0_121-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.121-b13, mixed mode)

C:\Users\MICRO>java -jar contextnet-2.7.jar 127.0.0.1 5500 OpenSplice
```

Figure A.3: Command line that deploys gateway locally

A.3.2

Mobile Hub Setup

The Mobile Hub (M-Hub) is a general-purpose middleware that enables mobile personal devices (Android smartphones and tablets) to become the propagator nodes (i.e. gateways to the Internet) for the simpler IoT objects or Mobile Objects (M-OBJ) (sensors/actuators) with only short-range WPAN interfaces. It provides context information such as current local time and/or the (approximate) location to the data obtained from the M-OBJs to which it is connected. The M-Hub extends the ContextNet with IoT access to smart devices (with WPAN interfaces) [26].

It is possible to use M-Hub Android application package (figure A.4) that is provided by the developers of the middleware in the Mobile Hub project page [26], which allows the user to easily start using it and enable your device to collect M-OBJ data. With minimal and simple parameter configuration, such as the IP address and the port of a Gateway that should be connected (as seen in the first image of figure A.4), any data from all nearby sensors of IoT devices can be sent to the server through ContextNet's SDDL. The user can also configure other parameters, for instance: a Bluetooth LE strength threshold, time intervals between BLE scans, and enabling or disabling of specific services (as shown in the top images of figure A.4). Data collected can be visualized in the Mobile-Hub app's (as displayed in the bottom images of figure A.4). It is important to note that the M-Hub application requires Android 4.3 version or newer to work [26].

In this project, where the functionality of scanning surroundings of the mobile device in order to find sensors and actuators is a part of the gameplay, we wanted to let the player decide when to start and stop searching for M-OBJs, as opposed to enable the scanning throughout the whole execution of the game. This user-driven device search we took not only fits within the context of games, but also saves battery of the user device, as it not only prevents the device to be scanning for M-OBJs all the time, but it also removes the necessity of having the BLE interface turned on while playing the game, even when is not needed. The result is that when the player wants to scan, the Bluetooth will automatically be enabled, and when the user wants to stop, the Bluetooth will be disabled.

Thus, we deviated from using the original Mobile Hub application, as it controls the scanning with its own on and off button. Instead, we wanted to control the scanning from inside the game, facilitating the scanning for the player and not having to switch between mobile applications. To achieve that we used The Mobile Hub API, that provides all the existing services that

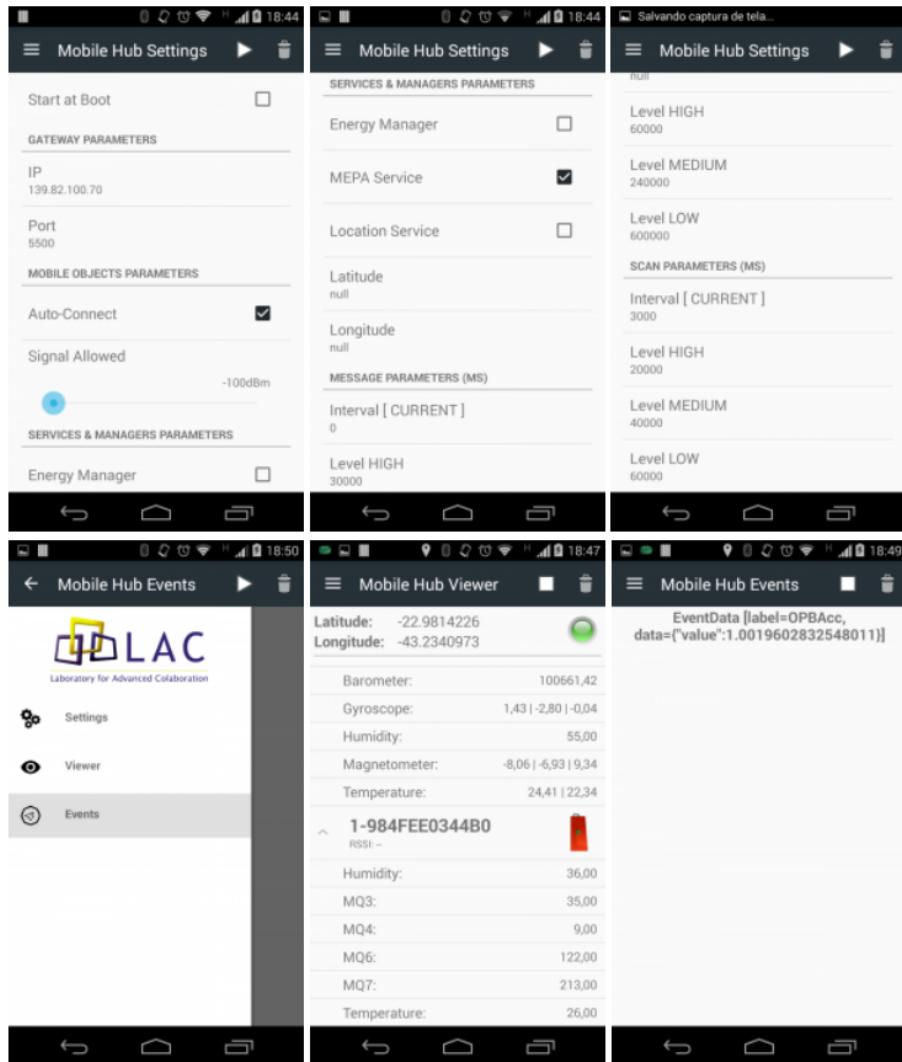


Figure A.4: Mobile Hub application

compose M-Hub. We used the service called S2PA, which is responsible for monitoring and discovering M-OBJs.

Once you have the Mobile Hub API [26] in your project, you will need to be able to use the S2PA service of M-Hub. Thus, you must declare it in the Android manifest file as shown in figure A.5. For guidance, the complete Android manifest file used in this project is available in the appendix section.

```

33    <!-- Begin Services -->
34
35    <service android:name="br.pucrio.inf.lac.mhub.services.S2PAService" />
36    <service android:name="br.pucrio.inf.lac.mhub.services.ConnectionService" />
37    <service android:name="br.pucrio.inf.lac.mhub.services.LocationService" />
38    <service android:name="br.pucrio.inf.lac.mhub.services.AdaptationService" />
39    <service android:name="br.pucrio.inf.lac.mhub.services.MEPAService" />
40
41    <!-- End Services -->
```

Figure A.5: Android manifest file services declaration

Each declaration in figure A.5 refers to a service present in the Mobile

Hub API. The name that references each service is nothing more than the package path concatenated with the service class name.

The S2PA service will be in charge for discovering and for monitoring data from the M-OBJs, and will start to do so at the moment it is instantiated, but the data retrieved will not be automatically sent to the server, since this is not part of the S2PA service job. In this project, we used our already implemented client-server communication that was created through SDDL to send any data retrieved via the S2PA service data. For that to work, an interface between the Android native code and the LibGDX framework had to be developed. The details of this interface as well as the implementation of this process is discussed in chapter 5.

Finally, the Mobile Hub will, in some cases, require some configurations for the IDE used. In our case, the Android Studio studio.vmoptions file should mirror the parameters shown in figure A.6.

```
-server
-Xms512m
-Xmx1024m
-XX:MaxPermSize=1024m
-XX:ReservedCodeCacheSize=150m
-XX:+UseConcMarkSweepGC
-XX:SoftRefLRUPolicyMSPerMB=50
-ea
-Djna.nosys=true
-Djna.boot.library.path=

-Djna.debug_load=true
-Djna.debug_load.jna=true
-Dsun.io.useCanonCaches=false
-Djava.net.preferIPv4Stack=true
-Dawt.useSystemAAFontSettings=lcd
```

Figure A.6: Android Studio studio.vmoptions file

A.4 Gradle Setup

LibGDX framework generated project creates a Gradle-ready project, so gradle can be used to build the project in different IDEs. Following the same protocol, we included any external libraries needed to run the project in

the Gradle build, including ContextNet's JAR file. It is also possible to use dependencies that are stored in remote repositories, as illustrated in figure A.7.

```

140     dependencies {
141
142         compile 'com.google.dagger:dagger:2.0'
143         compile fileTree(include: '*.jar', dir: 'libs')
144         compile 'com.android.support:support-v4:25.1.0'
145         compile 'com.android.support:multidex:1.0.1'
146     }

```

Figure A.7: Gradle local and remote dependencies

Figure A.7 shows the including of three different remote dependencies, but also shows how to compile local dependencies through the parameter "fileTree" which in this case includes the compilation of any JAR file stored in the local libs directory. The problem is that the generated LibGDX project consists of two separated sub-projects, one that works as a normal Android project that is named "android", and another one where the framework will work, named "core". To build these sub-projects as one bigger project, the Gradle setup ends up getting a little bit more complicated, resulting in a project with multiple Gradle build files and libs folders.

To guide on where each local dependency should be stored, as well as in which Gradle build file should the dependencies be declared, LibGDX provides an extensive document detailing dependency management [47]. Even with the provided guide, it is not trivial to make the LibGDX framework and ContextNet middleware to work all together, so in order to assist future projects we provide in the appendix section the fully working three gradle files, as well as displaying in the figure (A.8) the disposition of the libs folder and the necessary libraries.

One last remark that should be made about Gradle is that the whole project was based on an older version of Gradle, in consequence of the LibGDX generated project Gradle version. If the newer version is used, some keywords and structures may vary, so a compatibility work may be needed.

A.5 Database Setup

To manage the server database, we chose to use phpMyAdmin. It is a free software tool written in PHP, intended to handle the management of MySQL databases over the Web. phpMyAdmin supports a wide range of operations on MySQL. Frequently used operations (managing databases, tables, columns, relations, indexes, users, permissions, etc) can be performed via the user

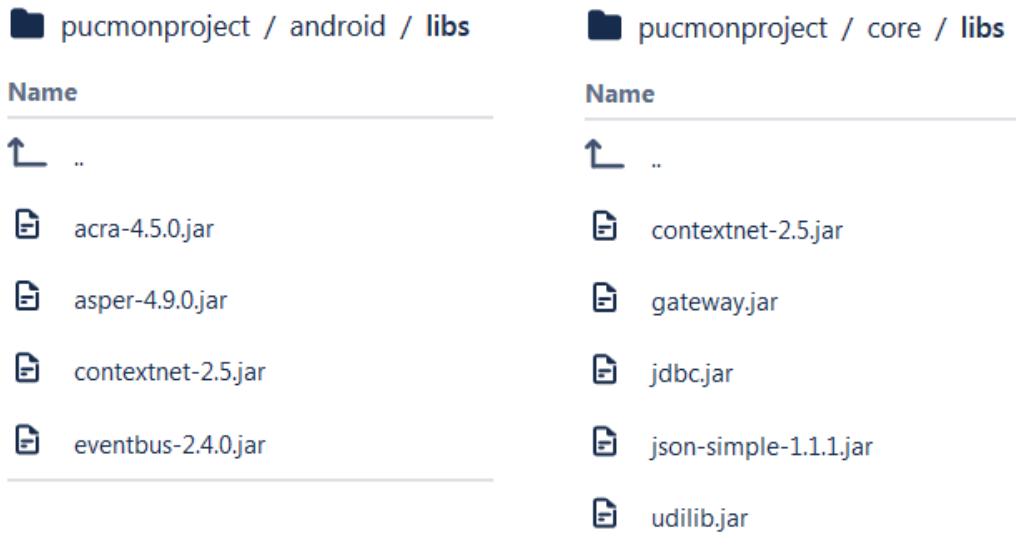


Figure A.8: Android libs folder (left) and core libs folder (right)

interface, while one still has the ability to directly execute any SQL statement [48].

It is usually ranked as one of the best among MySQL administration tool [49], and is our go-to option for database administration. We used XAMPP as web server (figure A.9), which comes with phpMyAdmin bundled in the installation for database administration.

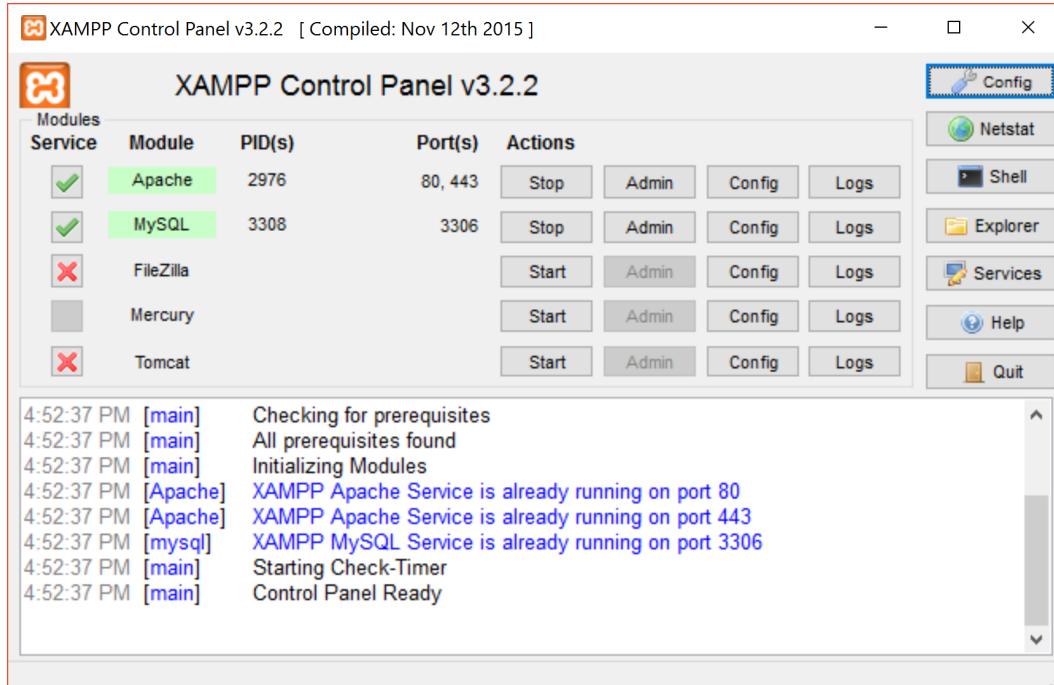


Figure A.9: XAMPP web server with Apache and MySQL services on

Newer versions of XAMPP comes with MariaDB instead of MySQL, but is possible to replace it with MySQL [50], although we used an older version

that comes with MySQL. Once XAMPP is installed, you will only need to start Apache and MySQL services to be able to use and administrate your server database, as shown in figure A.9. You may need to make sure your ports are open for WAN access, but for local and LAN uses you are set up.

Depending if the access is within the server machine, a LAN or a WAN, the phpMyAdmin administration page (figure A.10) must be accessed distinctly, since the IP to access the machine varies. The page URL should be the IP to access the machine concatenated with "/phpmyadmin", so for local access within the machine, the URL would be "http://localhost/phpmyadmin" or "http://127.0.0.1/phpmyadmin". It is also strongly advisable to create a password for phpMyAdmin, ensuring the security of the data.

The screenshot shows the phpMyAdmin interface for a MySQL database on a local host. The top navigation bar includes links for Databases, SQL, Status, Processes, Privileges, Export, Import, and More. On the left, a sidebar lists databases: information_schema, mysql, performance_schema, and phpmyadmin. The main content area is divided into several panels:

- General Settings:** Includes a 'Change password' link and a dropdown for 'MySQL connection collation' set to 'utf8_general_ci'.
- Appearance Settings:** Includes a 'Language' dropdown set to 'English', a 'Theme / Style' dropdown set to 'pmahomme', and a 'Font size' dropdown set to '82%'.
- MySQL:** Displays server information: Localhost via UNIX socket, Server version: 5.5.40-0+wheezy1, Protocol version: 10, User: root@localhost, and MySQL charset: UTF-8 Unicode (utf8).
- Web server:** Displays Apache/2.2.22 (Debian), MySQL client version: 5.5.40, and PHP extension: mysqli.
- phpMyAdmin:** Includes a 'Version information' section with links to Documentation, Wiki, Official Homepage, Contribute, Get support, and List of changes.

Figure A.10: The phpMyAdmin page accessed locally

Naturally, phpMyAdmin allows the creation of multiple databases. Once a database is created, you may create multiple users with different access privileges for it, but we recommend that users with all privileges should be avoided for safety reasons.

To access and use the database created in the project, the following data will be necessary:

- The name of the database created for the project;
- The IP to access the database machine from the game server machine. As it probably is the same machine, 127.0.0.1 or LAN IP should be used;
- The port to access MySQL. Default is 3306, there is no need to change unless the port is being used for other applications;

- The username and the password of a user that has access to the database created for the project.

B

Android Manifest File

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mygdx.game" >

    <uses-sdk android:minSdkVersion="16"
        android:targetSdkVersion="25" />
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission
        android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission
        android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission
        android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission
        android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission
        android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission
        android:name="android.permission.RECEIVE_BOOT_COMPLETED" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/GdxTheme"
        >

        <activity
            android:name="com.mygdx.game.AndroidLauncher"
            android:label="@string/app_name"
            android:screenOrientation="portrait"
```

```
    android:configChanges="keyboard|keyboardHidden|orientation|screenSize">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category
            android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<!-- Begin Services -->

<service
    android:name="br.pucrio.inf.lac.mhub.services.S2PAService"
    />
<service
    android:name="br.pucrio.inf.lac.mhub.services.ConnectionService"
    />
<service
    android:name="br.pucrio.inf.lac.mhub.services.LocationService"
    />
<service
    android:name="br.pucrio.inf.lac.mhub.services.AdaptationService"
    />
<service
    android:name="br.pucrio.inf.lac.mhub.services.MEPAService"
    />

<!-- End Services -->

<!-- Begin Broadcast Receivers -->

<receiver
    android:name="br.pucrio.inf.lac.mhub.broadcastreceivers.OnBootReceiver"
    >
    <intent-filter>
        <action
            android:name="android.intent.action.BOOT_COMPLETED"
            />
    </intent-filter>
</receiver>

<receiver
    android:name="br.pucrio.inf.lac.mhub.broadcastreceivers.PowerReceiver"
```

```
>
<intent-filter>
    <action
        android:name="android.intent.action.ACTION_POWER_CONNECTED"
    />
    <action
        android:name="android.intent.action.ACTION_POWER_DISCONNECTED"
    />
</intent-filter>
</receiver>

<receiver
    android:name="br.pucrio.inf.lac.mhub.broadcastreceivers.ConnectivityReceiver"
>
<intent-filter>
    <action
        android:name="android.net.conn.CONNECTIVITY_CHANGE"
    />
</intent-filter>
</receiver>

<receiver
    android:name="br.pucrio.inf.lac.mhub.broadcastreceivers.BatteryReceiver"
/>

<!-- End Broadcast Receivers -->

</application>

```

C

Gradle Build Files

C.1 Root Folder Gradle File

```
/*****
*** root/build.gradle ***
*****/


buildscript {

    repositories {
        mavenLocal()
        mavenCentral()
        maven { url
            "https://oss.sonatype.org/content/repositories/snapshots/"
        }
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:2.3.3'
    }
}

allprojects {
    apply plugin: "eclipse"
    apply plugin: "idea"

    version = '1.0'
    ext {
        appName = "pucmon"
        gdxVersion = '1.9.6'
        roboVMVersion = '2.3.1'
    }
}
```

```
    box2DLightsVersion = '1.4'
    ashleyVersion = '1.7.0'
    aiVersion = '1.8.0'
}

repositories {
    mavenLocal()
    mavenCentral()
    maven { url
        "https://oss.sonatype.org/content/repositories/snapshots/"
    }
    maven { url
        "https://oss.sonatype.org/content/repositories/releases/"
    }
}
}

project(":android") {
    apply plugin: "android"

    configurations { natives }

    dependencies {
        compile project(":core")
        compile
            "com.badlogicgames.gdx:gdx-backend-android:$gdxVersion"
        natives
            "com.badlogicgames.gdx:gdx-platform:$gdxVersion:natives-armeabi"
        natives
            "com.badlogicgames.gdx:gdx-platform:$gdxVersion:natives-armeabi-v7a"
        natives
            "com.badlogicgames.gdx:gdx-platform:$gdxVersion:natives-arm64-v8a"
        natives
            "com.badlogicgames.gdx:gdx-platform:$gdxVersion:natives-x86"
        natives
            "com.badlogicgames.gdx:gdx-platform:$gdxVersion:natives-x86_64"
        compile "com.badlogicgames.gdx:gdx-box2d:$gdxVersion"
        natives
            "com.badlogicgames.gdx:gdx-box2d-platform:$gdxVersion:natives-armeabi"
        natives
            "com.badlogicgames.gdx:gdx-box2d-platform:$gdxVersion:natives-armeabi-v7a"
}
```

```

natives
    "com.badlogicgames.gdx:gdx-box2d-platform:$gdxVersion:natives-arm64-v8a"
natives
    "com.badlogicgames.gdx:gdx-box2d-platform:$gdxVersion:natives-x86"
natives
    "com.badlogicgames.gdx:gdx-box2d-platform:$gdxVersion:natives-x86_64"

}

}

project(":core") {
    apply plugin: "java"

    dependencies {
        compile "com.badlogicgames.gdx:gdx:$gdxVersion"
        compile "com.badlogicgames.gdx:gdx-box2d:$gdxVersion"
        compile fileTree(dir: 'libs', include: '*.jar')

    }
}

tasks.eclipse.doLast {
    delete ".project"
}

```

C.2 Android Folder Gradle File

```

/*****
*** root/android/build.gradle ***
*****/


android {
    buildToolsVersion "26.0.1"
    compileSdkVersion 25
    sourceSets {
        main {
            manifest.srcFile 'AndroidManifest.xml'
            java.srcDirs = ['src']

```

```

        aidl.srcDirs = ['src']
        renderscript.srcDirs = ['src']
        res.srcDirs = ['res']
        assets.srcDirs = ['assets']
        jniLibs.srcDirs = ['libs']
    }

    instrumentTest.setRoot('tests')
}

packagingOptions {
    exclude 'META-INF/robovm/ios/robovm.xml'
}

defaultConfig {
    applicationId "com.mygdx.game"
    minSdkVersion 16
    targetSdkVersion 25
    versionCode 1
    versionName "1.0"
    // Enabling multidex support.
    multiDexEnabled true
}
dexOptions {
    jumboMode true
    javaMaxHeapSize "4g"
}
buildTypes {
    release {
        minifyEnabled true
        proguardFiles {
            getDefaultProguardFile('proguard-android.txt'),
            'proguard-rules.pro'
        }
    }
}

// called every time gradle gets executed, takes the native
// dependencies of
// the natives configuration, and extracts them to the proper libs/
// folders
// so they get packed with the APK.
task copyAndroidNatives() {

```

```
file("libs/armeabi/").mkdirs();
file("libs/armeabi-v7a/").mkdirs();
file("libs/arm64-v8a/").mkdirs();
file("libs/x86_64/").mkdirs();
file("libs/x86/").mkdirs();

configurations.natives.files.each { jar ->
    def outputDir = null
    if (jar.name.endsWith("natives-arm64-v8a.jar")) outputDir =
        file("libs/arm64-v8a")
    if (jar.name.endsWith("natives-armeabi-v7a.jar")) outputDir =
        file("libs/armeabi-v7a")
    if (jar.name.endsWith("natives-armeabi.jar")) outputDir =
        file("libs/armeabi")
    if (jar.name.endsWith("natives-x86_64.jar")) outputDir =
        file("libs/x86_64")
    if (jar.name.endsWith("natives-x86.jar")) outputDir =
        file("libs/x86")
    if (outputDir != null) {
        copy {
            from zipTree(jar)
            into outputDir
            include "*.so"
        }
    }
}
}

task run(type: Exec) {
    def path
    def localProperties = project.file("../local.properties")
    if (localProperties.exists()) {
        Properties properties = new Properties()
        localProperties.withInputStream { instr ->
            properties.load(instr)
        }
        def sdkDir = properties.getProperty('sdk.dir')
        if (sdkDir) {
            path = sdkDir
        } else {
            path = "$System.env.ANDROID_HOME"
        }
    }
}
```

```

} else {
    path = "$System.env.ANDROID_HOME"
}

def adb = path + "/platform-tools/adb"
commandLine "$adb", 'shell', 'am', 'start', '-n',
    'com.mygdx.game/com.mygdx.game.AndroidLauncher'
}

// sets up the Android Eclipse project, using the old Ant based
// build.

eclipse {
    // need to specify Java source sets explicitly, SpringSource
    // Gradle Eclipse plugin
    // ignores any nodes added in classpath.file.withXml
    sourceSets {
        main {
            java.srcDirs "src", 'gen'
        }
    }
}

jdt {
    sourceCompatibility = 1.6
    targetCompatibility = 1.6
}

classpath {
    plusConfigurations += [project.configurations.compile]
    containers 'com.android.ide.eclipse.adt.ANDROID_FRAMEWORK',
        'com.android.ide.eclipse.adt.LIBRARIES'
}

project {
    name = appName + "-android"
    natures 'com.android.ide.eclipse.adt.AndroidNature'
    buildCommands.clear();
    buildCommand
        "com.android.ide.eclipse.adt.ResourceManagerBuilder"
    buildCommand "com.android.ide.eclipse.adt.PreCompilerBuilder"
    buildCommand "org.eclipse.jdt.core.javabuilder"
    buildCommand "com.android.ide.eclipse.adt.ApkBuilder"
}

```

```

}

// sets up the Android Idea project, using the old Ant based build.
idea {
    module {
        sourceDirs += file("src");
        scopes = [COMPILE: [plus: [project.configurations.compile]]]

        iml {
            withXml {
                def node = it.asNode()
                def builder = NodeBuilder.newInstance();
                builder.current = node;
                builder.component(name: "FacetManager") {
                    facet(type: "android", name: "Android") {
                        configuration {
                            option(name: "UPDATE_PROPERTY_FILES",
                                   value: "true")
                        }
                    }
                }
            }
        }
    }
}

dependencies {

    compile 'com.google.dagger:dagger:2.0'
    compile fileTree(include: '*.jar', dir: 'libs')
    compile 'com.android.support:support-v4:25.1.0'
    compile 'com.android.support:multidex:1.0.1'
}

```

C.3 Core Folder Gradle File

```

*****
*** root/core/build.gradle ***
*****


apply plugin: "java"

```

```
sourceCompatibility = 1.6
[compileJava, compileTestJava]*.options*.encoding = 'UTF-8'
sourceSets.main.java.srcDirs = ["src/"]
eclipse.project {
    name = appName + "-core"
}
dependencies {
```

D

SQL Schema File

```
DROP TABLE IF EXISTS 'player_equipments';
DROP TABLE IF EXISTS 'player_items';
DROP TABLE IF EXISTS 'players';
DROP TABLE IF EXISTS 'accounts';
DROP TABLE IF EXISTS 'market';
DROP TABLE IF EXISTS 'mission_storage';
DROP TABLE IF EXISTS 'sensors';
DROP TABLE IF EXISTS 'sensors_input';

CREATE TABLE 'accounts'
(
    'id' INT NOT NULL AUTO_INCREMENT,
    'name' VARCHAR(32) NOT NULL DEFAULT '',
    'password' VARCHAR(255) NOT NULL/* VARCHAR(32) NOT NULL COMMENT
        'MD5' *//* VARCHAR(40) NOT NULL COMMENT 'SHA1' */,
    PRIMARY KEY ('id'), UNIQUE ('name')
) ENGINE = InnoDB;

INSERT INTO 'accounts' VALUES (1, '1', '1');

CREATE TABLE 'players'
(
    'id' INT NOT NULL AUTO_INCREMENT,
    'name' VARCHAR(255) NOT NULL,
    'world_map' INT NOT NULL DEFAULT 0,
    'account_id' INT NOT NULL DEFAULT 0,
    'level' INT NOT NULL DEFAULT 1,
    'experience' BIGINT NOT NULL DEFAULT 0,
    'posx' INT NOT NULL DEFAULT 0,
    'posy' INT NOT NULL DEFAULT 0,
    'gold' BIGINT NOT NULL DEFAULT 0,
    'first_login' BOOLEAN NOT NULL DEFAULT TRUE,
    'online' BOOLEAN NOT NULL DEFAULT FALSE,
```

```

PRIMARY KEY ('id'), UNIQUE ('name'),
KEY ('account_id'),
FOREIGN KEY ('account_id') REFERENCES 'accounts'('id') ON DELETE
CASCADE
) ENGINE = InnoDB;

INSERT INTO 'players' VALUES (1, 'Jotun', 0, 1, 1, 0, 0, 0, 0, true,
false);

CREATE TABLE 'player_items'
(
'id' INT NOT NULL AUTO_INCREMENT,
'uid' INT NOT NULL,
'player_id' INT NOT NULL DEFAULT 0,
'level' INT NOT NULL DEFAULT 0,
'page' INT NOT NULL DEFAULT 0,
'idxi' INT NOT NULL DEFAULT 0,
'idxj' INT NOT NULL DEFAULT 0,
PRIMARY KEY ('id'),
KEY ('player_id'),
FOREIGN KEY ('player_id') REFERENCES 'players'('id') ON DELETE
CASCADE
) ENGINE = InnoDB;

INSERT INTO 'player_items' VALUES (1, 3000, 1, 1, 1, 0, 0);

CREATE TABLE 'player_equipments'
(
'id' INT NOT NULL AUTO_INCREMENT,
'uid' INT NOT NULL,
'player_id' INT NOT NULL DEFAULT 0,
'level' INT NOT NULL DEFAULT 0,
'slot' INT NOT NULL DEFAULT 0,
PRIMARY KEY ('id'),
KEY ('player_id'),
FOREIGN KEY ('player_id') REFERENCES 'players'('id') ON DELETE
CASCADE
) ENGINE = InnoDB;

INSERT INTO 'player_equipments' VALUES (1, 3000, 1, 1, 1);

```

```

CREATE TABLE `market`
(
    `id` INT NOT NULL AUTO_INCREMENT,
    `uid` INT NOT NULL,
    `player_id` INT NOT NULL DEFAULT 0,
    `level` INT NOT NULL DEFAULT 0,
    `price` BIGINT NOT NULL DEFAULT 0,
    `quality` INT NOT NULL DEFAULT 0,
    `sold` BOOLEAN NOT NULL DEFAULT FALSE,
    PRIMARY KEY (`id`),
    KEY (`player_id`),
    FOREIGN KEY (`player_id`) REFERENCES `players`(`id`) ON DELETE CASCADE
) ENGINE = InnoDB;

INSERT INTO `market` VALUES (1, 7000, 1, 10, 520, 3, false);

CREATE TABLE `mission_storage`
(
    `mission_id` INT NOT NULL DEFAULT 0,
    `player_id` INT NOT NULL DEFAULT 0,
    `timestamp` timestamp NOT NULL DEFAULT now(),
    PRIMARY KEY (`mission_id`, `player_id`),
    KEY (`player_id`),
    FOREIGN KEY (`player_id`) REFERENCES `players`(`id`) ON DELETE CASCADE
) ENGINE = InnoDB;

INSERT INTO `mission_storage` VALUES (0, 1, NULL);

CREATE TABLE `sensors`
(
    `sensor_id` VARCHAR(255) NOT NULL,
    `sensor_type` VARCHAR(255) NOT NULL DEFAULT 'mission',
    `mission_id` INT,
    `thumbnail_id` INT,
    `code` VARCHAR(255),
    `n_inputs` INT NOT NULL DEFAULT 0,
    `input_type` VARCHAR(255),
    PRIMARY KEY (`sensor_id`)
) ENGINE = InnoDB;

```

```
INSERT INTO `sensors` VALUES ('1-F45EAB2755CC', 'mission', 0, 0,
    'RGBR', 0, NULL);
INSERT INTO `sensors` VALUES ('1-34B1F7D508B2', 'mission', 1, 1,
    NULL, 2, 'integer');

CREATE TABLE `sensors_input`
(
    `id` INT NOT NULL AUTO_INCREMENT,
    `sensor_id` VARCHAR(255) NOT NULL,
    `input_type` VARCHAR(255) NOT NULL,
    `input_name` VARCHAR(255) NOT NULL,
    `input_data` VARCHAR(255) NOT NULL,
    `timestamp` timestamp NOT NULL DEFAULT now(),
    PRIMARY KEY (`id`),
    KEY `sensor_id`(`sensor_id`),
    FOREIGN KEY (`sensor_id`) REFERENCES `sensors`(`sensor_id`) ON
        DELETE CASCADE
) ENGINE = InnoDB;
```

E Data Source Factory

```
/**  
 * Factory of MySqlDataSource object  
 * that communicates with server database  
 *  
 * @author Pedro Sampaio  
 * @since 1.4  
 */  
  
public class DataSourceFactory {  
  
    public static DataSource getMySQLDataSource() {  
        Properties props = new Properties();  
        FileInputStream fis = null;  
        MysqlDataSource mysqlDS = null;  
        String path = "sql/db.properties";  
        try {  
            fis = new FileInputStream(path);  
            props.load(fis);  
            mysqlDS = new MysqlDataSource();  
            mysqlDS.setURL(props.getProperty("MYSQL_DB_URL"));  
            mysqlDS.setUser(props.getProperty("MYSQL_DB_USERNAME"));  
            mysqlDS.setPassword(props.getProperty("MYSQL_DB_PASSWORD"));  
        } catch (IOException e) {  
            System.err.println("Could not find db properties file:  
                "+path);  
            e.printStackTrace();  
        }  
        return mysqlDS;  
    }  
}
```
