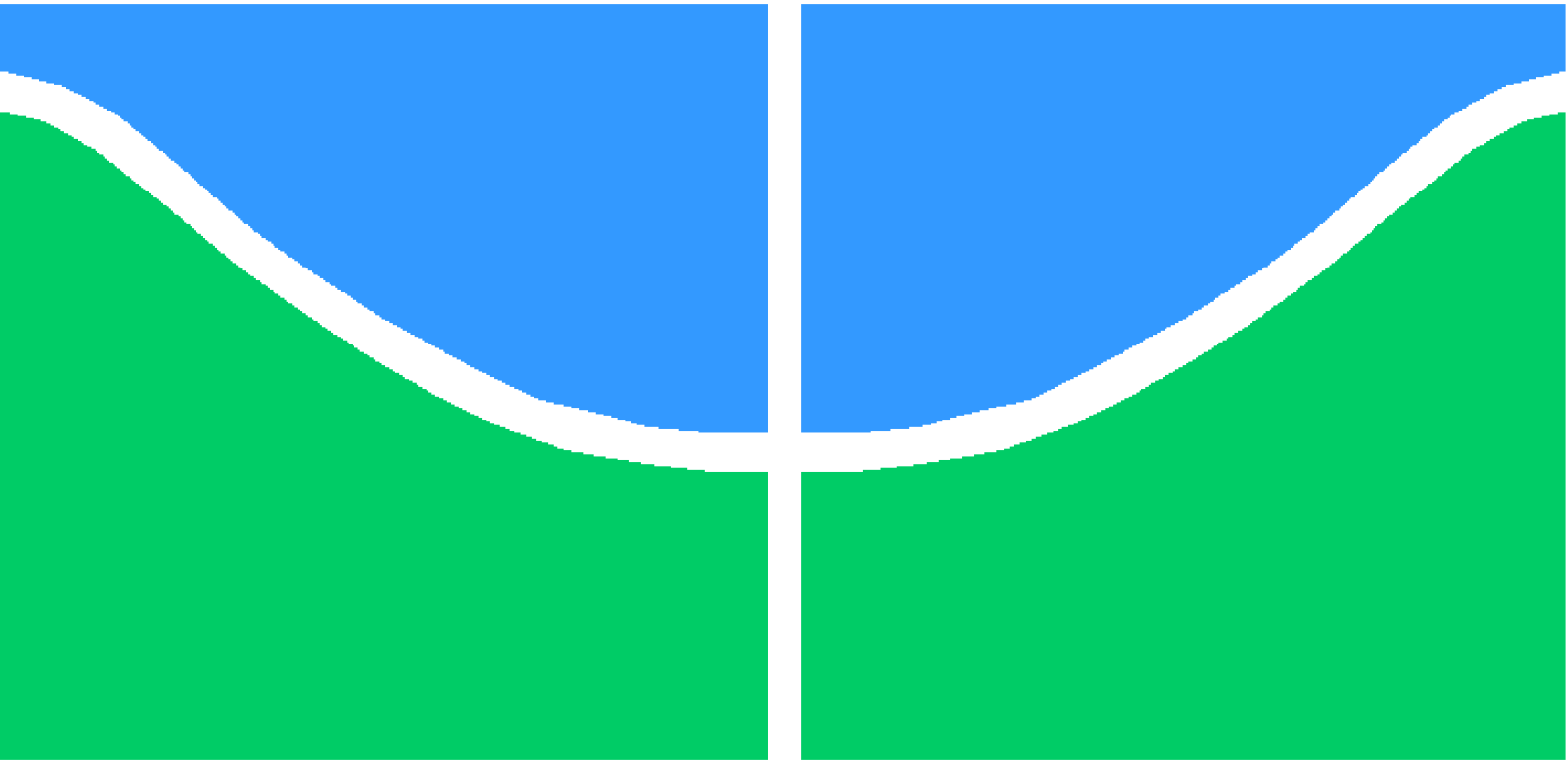

Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia de Software

UnB Games: A collaborative project

Autor: Parley Pacheco Martins
Orientador: Prof. Dr. Edson Alves Da Costa Júnior

Brasília, DF
2017

Parley Pacheco Martins

UnB Games: A collaborative project

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Supervisor: Prof. Dr. Edson Alves Da Costa Júnior

Co-supervisor: Prof. Matheus de Sousa Faria

Brasília, DF

2017

Parley Pacheco Martins
UnB Games: A collaborative project/ Parley Pacheco Martins. – Brasília, DF, 2017-
60 p. : il. (algumas color.) ; 30 cm.

Supervisor: Prof. Dr. Edson Alves Da Costa Júnior

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2017.

1. packaging. 2. game. I. Prof. Dr. Edson Alves Da Costa Júnior. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. UnB Games: A collaborative project

CDU 02:141:005.6

Parley Pacheco Martins

UnB Games: A collaborative project

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, :

Prof. Dr. Edson Alves Da Costa Júnior
Supervisor

Guest 1

Guest 2

Brasília, DF
2017

Acknowledgements

First of all, thank you to my dear family! I couldn't have reached this far without you! My mom, Gláucia Maria Pacheco Terra, and my dad, Paulo Orlando Martins, whom have loved me since my birth and taught me how to be a better person and fight difficulties in life. To my brother Sam Pacheco Martins and sister Élida Pacheco Martins, whom I love so much, despite all the worries they give me. To my extended family, uncles, aunts, cousins, that are so many to name each, but have seen my potential even when I couldn't see it.Thank you sou much! Your faith in me makes me go further than I ever thought I could!

I also want to thank with all the warmth of my heart the friends who have been with me in this journey and helped me through this stressful period of life, either here in Brazil or in my dear Canada. Huge thanks to one of my favorite people in the world Mateus Medeiros Furquim Mendonça and his entire family, Elis, Geovane and Tiago, for helping me keeping my sanity with all the laughs and games, and also for giving me a place to study. I couldn't have done half of this work if you hadn't welcomed and sheltered me this semester. I love you all!

Special thanks to my supervisor, Professor Edson Alves da Costa Junior, who saw potential on a 16 years old boy and never stopped believing in me ever since. Thanks for guiding me through this work.

"There are some things you can't share without ending up liking each other and knocking out a twelve-foot mountain troll is one of them."
(Harry Potter and the Philosopher's Stone)

Resumo

Jogos desenvolvidos nas universidades não possuem muito reconhecimento ou suporte. Usuários raramente têm a chance de jogar tais jogos ou dar algum *feedback*, como críticas, elogios e reportar problemas, ao desenvolvedor sobre qualquer versão desses jogos. A maioria das pessoas nem sabe que jogos são feitos em salas de aula. Este projeto tem como objetivo tornar esses jogos disponíveis para o público, através do desenvolvimento de uma plataforma online. Todo o trabalho realizado numa universidade, especialmente pública, deve ser acessível a toda a sociedade, desde a concepção até a implementação, por isto, este documento descreve como este trabalho será feito, criando a plataforma para o compartilhamento desses jogos. A plataforma também possibilitará a geração de pacotes, para que o usuário consiga instalá-los sem dificuldade.

Palavras-chaves: jogos. desenvolvimento. plataforma. empacotamento.

Abstract

Games developed in the University don't have much recognition or support. Users don't usually get to play them or give a feedback about any version of any of them, either good, bad or bug reports. Most people don't even know that games are created in classes. This project aims to make these games available to people, by developing an on-line platform. Everything created in the university, especially a public one, should be accessible to the society, since its conception to its implementation. Because of that, this document outlines how this work will be achieved, by creating a platform to upload the developed games. It will also provide the building of packages to simplify the installation process for the user.

Key-words: games. development. platform. packaging.

List of Figures

List of Tables

Listings

List of abbreviations and acronyms

SDL	Simple DirectMedia Layer version 1
SDL2	Simple DirectMedia Layer version 2
API	Application Program Interface
GUI	Graphical User Interface
VM	Virtual Machine
OS	Operating System
dpkg	Debian Package Management System
rpm	RPM Package Manager
pacman	Pacman Package Manager
RUP	Rational Unified Process
XP	eXtreming Programming
FHS	Filesystem Hierarchy Standard
FGA	*Faculdade UnB Gama*
MDS	*Métodos de Desenvolvimento de Software*
GPP	*Gestão de Portfolios e Projetos*
PMBoK	Project Management Body of Knowledge
LAN	Local Area Network
indie	Independent
GPL	GNU General Public License
VS	Visual Studio

Contents

Introduction

Games are known to provide several benefits to the players. It may be enjoying a good story, developing new abilities and skills, bonding with friends or just relaxing after a big rushed day. Independent game developers have to struggle to achieve any of these goals, because it's so much harder for people to see their games.

There are some courses taught here in the *Universidade de Brasília* (like *Introdução ao Desenvolvimento de Jogos* at the campus *Darcy Ribeiro*; and *Introdução aos Jogos Eletrônicos* at the campus Gama) that have the goal to teach students to develop games. The students that take these have the opportunity to learn how to create a game from scratch. Several of these students wish to continue working on game development after their graduation.

The games developed in those courses usually have a good story and are good to play with, however they are never seen outside the courses because there's nowhere to put them after they are done. People also have the tendency to relate things that are done inside the classes to things that have no use in *real life*, therefore expandable.

This project was created to give visibility to these games and developers and to show the work that has and will be done in this University concerning game development.

Goals

The main goal of this project is to create an on-line platform to host the games developed in the courses of this University. The secondary goals are the following:

- allow users to download, run and distribute these games in any operating system they have;
- let the students of these courses upload their source codes and have the respective installers and packages available for the public;
- build packages to games that don't have one.

Work Structure

This document is divided in chapters. Chapter 1 explains some basic concepts for the reader. Chapter 2 given an overview of the tasks to be done and how they were achieved. Chapter 3 shows the partial results the project had so far, as well as the issues

with those results. Chapter **??** describes the next steps needed to achieve the main goal, with a brief schedule containing the estimated time to complete the tasks.

1 Basic Concepts

This chapter gives an overview of some basic concepts needed by the reader to understand this work. It starts talking about games and the SDL library, then talks a little about the GNU/Linux Filesystem, that helps developers to know where their binaries and other files should go on the user's system. In the end, there are brief words on repositories and packages.

1.1 Games

Games have been a part of human development since their early childhood and have been part of history in its most basic ways (BETHKE, 2003). Providing a fun time, bonding with friends and learning new skills are some common goals of games. They consist of interacting with other people (or computer) or just with the game structure itself, following the rules to achieve a goal.

They can take several formats, like board and card games, for example. Each form has unique strategies to win. To illustrate that, take the two cited cases: board games usually divide the user space into sectors, and everything is related to which sectors you are in and how you control them; card games, however, rely on the symbols and possible combinations of them (CRAWFORD, 1984). To win the former type, a player has to understand the cost to acquire/leave sectors and plan accordingly, while on the latter, one needs to watch their symbols and try to get the best combination out of them.

Since computers were invented, they completely changed the gaming world. New kinds of games, like *first person shooter*, and *tower defense*, were created and made accessible, while it became possible to play virtually the ones that required a physical board or a lot of people. With the Internet, it grew even more comfortable to own and play different games. It's also possible to play any type of game with anyone in the world.

Because computer games are software with audio, art, and gameplay, they should follow a software development method, anyone chosen by the team. This is something that most game developers avoid because they see their work as pure art (BETHKE, 2003). Although that is probably true, a game has everything a "normal software" has and more, therefore requiring a known development process or method. Using software engineering techniques (adapted to their needs, naturally) will result in a better game and better interaction with the final user (PRESSMAN, 2010).

1.2 SDL

Digital games have many things happening at once. There is sound playing; they must be able to receive and respond to inputs from the player, coming from different sources sometimes, and, while all this is happening, they must also keep rendering the scenario and show the statistics of the user. To simplify that, developers use several libraries in their source codes, one of the most popular being SDL.

Simple DirectMedia Layer (SDL) is a library that helps developers by creating cross-platform APIs to make easier handling video, input, audio, threads. It's used in several games available in big platforms like *Steam* and *Humble Bundle* (SDL, 2017). To be fully integrated with the developer's code, a few files are needed during the compilation: the headers, that contains definitions of functions and structures; and the library itself, that contains the binaries that will run with the main code, and may be static and shared (MITCHELL, 2013).

A shared library is one that can be used in multiple programs. It provides common code that is reusable and can be linked to the developer's code at the running time. On GNU/Linux systems they have the *.so* file extension, while on Windows they have *.dll* (CAMPBELL, 2009). In this case, the library code is not merged to the main code, resulting in a smaller binary for the developer. It's required to have the library installed on the user's system, though.

The static library is compiled against the main source code, and it's merged with it. Instead of being a dependency on the user's system, it's now a part of the distributed version of the software, resulting in a more prominent binary. The new license on SDL2, *zlib*[1], allows users to use SDL as a static library; however, they are not encouraged to, because that wouldn't provide several things the user might need. For example, security updates that come on the new patches, wouldn't be available to a game that has SDL built into it (GORDON, 2017).

1.3 Repository

Game development, as has been said, demands special care with the source code. Like any software, when a bug is accidentally inserted, there should be an easy way to return to a previous state, where that didn't happen. The solution to this problem is using a repository for the source code.

According to the Merriam-Webster Dictionary (2017), a repository is "a place, room or container where something is deposited." A software repository is a computer,

[1] The text of this license can be found at <https://www.zlib.net/zlib_license.html>

directory or server that stores all the source code for that software project. This is usually available on the Internet, but it can also be local to the developers.

Repositories are also related to the version control of the source code being produced. The definition of version control is "a system that records changes to a file or set of files over time so that you can recall specific versions later" (LOELIGER; MCCULLOUGH, 2012). This allows the user to compare versions, to check updates, see who introduced (or removed) an issue and to rollback to previous versions of the system (CHACON; STRAUB, 2014). The goal is to make it easy to return to states that were working, even after changes are made after a long time.

Modern version control systems allow developers to work on a distributed basis and to parallel their tasks, with the ability of *branching* the repository. Those *branches* are separated lines of development, that won't mess with the main one until they are merged (WESTBY, 2015). This feature lets developers create and test new changes before submitting them to the project's stable line of work, without affecting the final product.

1.4 Linux Filesystem Hierarchy Standard

When installing a game, it must go somewhere in the filesystem of the user. For games developed to run in the GNU/Linux environments, they should follow the patterns found in FHS. The Filesystem Hierarchy Standard (FHS) was proposed on February 14, 1994, as an effort to rebuild the file and directory structure of Linux and, later, all Unix-like systems. It helps developers and users to predict the location of existing and new files on the system, by proposing how minimum files, directories and guiding principles (BANDEL; NAPIER, 2001).

The Hierarchy starts defining types of files that can exist in a system. Whenever files differ in this classification, they should be located in different parts of the system: *shareable* files are the ones that can be accessed from a remote host, while *unshareable* are files that have to be on the same machine to be obtained. *Static* files are the ones that aren't supposed to be changed without administrator privileges, whereas *variable* ones can be altered by regular users (BANDEL; NAPIER, 2001)

The root filesystem is defined then: this should be as small as possible, and it should contain all the required files to boot, reset or repair the system. It must have the directories specified in Table 1 and installed software should never create new directories on this filesystem (ALLBERY et al., 2015).

From the directories in Table 1, "`/usr`, `/opt` and `/var` are designed such that they may be located on other partitions or filesystems." (ALLBERY et al., 2015). The `/usr` hierarchy should include shareable data, that means that every information host-

Table 1 – Directories on the Hierarchy (ALLBERY et al., 2015)

Directory	Description
`bin`	Essential command binaries
`boot`	Static files of the boot loader
`dev`	Device files
`etc`	Host-specific system configuration
`lib`	Essential shared libraries and kernel modules
`media`	Mount point for removable media
`mnt`	Mount point for mounting a filesystem temporarily
`opt`	Add-on application software packages
`run`	Data relevant to running processes
`sbin`	Essential system binaries
`srv`	Data for services provided by this system
`tmp`	Temporary files
`usr`	Secondary hierarchy
`var`	Variable data

specific should be placed in other directories. About the `/var` hierarchy, FHS specifies that "everything that once went into `/usr` that is written to during system operation (as opposed to installation and software maintenance) must be in `/var`." (ALLBERY et al., 2015).

The Hierarchy has some optionally defined places to put the binaries of the installed games, like `/usr/games`, or `/usr/local/games`. The difference between the two is that the former is where the package manager installs, while the other is usually where packages compiled locally are installed (TEAM, 2017). Variable data, as usual, should be inserted into the the `/var` filesystem, under `/var/games`.

1.5 Linux Packages

In computer science package can have multiple meanings, depending on the context being used. A GNU/Linux package means a bundle of files containing the required data to run an application, such as binaries and information about the package. Game packages behave precisely the same as any other software. To facilitate installing software, GNU/Linux has package managers.

Most Linux distributions have their own package managers. Each expects and handle different types of files, but all of them have the common goal of making the installation easier. They download the package, resolve dependencies, copy the needed binaries and execute any post- or pre-configuration required by the system to install a package (LINODE, 2017). For example, Debian has *dpkg*, Red Hat has *rpm* and Arch Linux has *pacman* as default package managers.

Another installing method is compiling from scratch. This may be very handy if the user is more advanced or the package is not in the package manager's repository. However, in this case, the user will have to manually handle dependencies, download, compile and do everything else the manager does.

1.6 Windows Registry

According to Wikipedia (2017a), "without a file system, information placed in a storage medium would be one large body of data with no way to tell where one piece of information stops and the next begins". Just like every Operating System, Windows has a Filesystem that handles how the archives are stored in this platform and also gives a naming convention for those files. On top of that there is the Windows Registry, that is a system-define database (MICROSOFT, 2017b) that provides configuration data for applications, user preferences, system configurations, for example (FISHER, 2017).

The Registry was created on early versions of Microsoft Windows to replace most of the `ini` files that contained system information and editing entries that don't relate to the developer's application may lead to a mal functioning system (MICROSOFT, 2017d). The Registry has a tree structure, with nodes called keys. Each subnode is a subkey and the data entries are called values, and one key (or subkey) can have as many values as it needds (MICROSOFT, 2017c).

1.7 Generating packages

Windows packages are slightly different than Linux's.

1.8 Related Work

There are several platforms to share and distribute games online. Amongst the most popular ones, there are Steam, GOG and Humble Bundle. They have thousands of games, including indies, with the great support of the gaming community. Some of them are described here as an inspiration for this work.

Steam was announced in 2002 and released in 2003. Valve saw the need that many games needed to run on up to date environment and decided to create a system that would target that issue (WIKIPEDIA, 2017c). The website has the purpose to be the store for the platform, while the system is installed on the player's machine and needed to play the games made available by them. Today, they have a vast community, cross-platform (Linux, Mac, mobile devices, and consoles) system with many games and extra content for them (STEAM, 2017).

GOG started out with the name *Good Old Games* trying to provide DRM free games to people. It was released in 2008 and has been active ever since (with a brief down period in September 2010). Since March 2012, it has been rebranded, and independent games have been added to their library (WIKIPEDIA, 2017b). They also have a system, like Steam, but this one is not required to run the games, it was built though to provide easy sharing and buying of games, among other things (GOG, 2017).

Humble Bundle is a platform that provides a bundle of games (books, software and other things) to the public at meager prices. Part of their profit is destined to charity (the buyer can also choose where their money goes), and they have already raised more than 98 million dollars for that purpose (BUNDLE, 2017). They also provide a store with regular prices and games.

Splitplay is a Brazilian platform specialized in indie games. They realized that several indie developers couldn't bring forth their games and decided to create a place where those would be publicized them in their own platform. Splitplay allows developers to send their games complete or incomplete (as a project), they can be free or paid. The site creators personally overview the submissions and don't charge developers (SPLITPLAY, 2017).

2 Methodology

This chapter explains how things were done within the duration of the whole project. Section 2.1 gives an overview of the whole project and its goals. Section 2.2 explains how the work was divided between all parties involved in the development of this project. Section 2.3 shows how and which games were selected for both parts of this work. Section 2.4 clarifies how the packaging template was created and its main parts. Section 2.5 illustrates how the platform was developed. Section 2.6 references the tools that were used to create and test everything the project has aimed to create.

2.1 Project Overview

The project had the main goal of creating a platform for all the games developed in the university's courses related to games. The games that will be available must have all their assets and required libraries in packages that run on some GNU/Linux, Windows and macOS systems.

In order to achieve this goal, the games developed in this *campus* of the University were cataloged and cloned into a main GitHub organization [1](whenever possible). A template system was created to package the files for each one of the contemplated Operating Systems.

The platform itself was developed while all the other activities took place, during the first half of this work. Some games were chosen to test the template, but its main use will be during the new development cycle inside the game courses.

2.2 Task Division

This project was totally collaborative, it depended and relied on different classes and courses. Because of that, during the first half of it, the work was divided among students and teachers, as illustrated in Figure 1.

Professor Edson and Mr. Faria were responsible for first cataloging the existing games. They remained as helpers in the packaging system and main stakeholders for the team that developed the website.

The team *Plataforma de Jogos UnB* from the courses *Métodos de Desenvolvimento de Software* (Software Development Methods) and *Gestão de Portfolios e Projetos* (Management of Portfolios and Projects) was in charge of creating the first version of the

[1] https://github.com/unbgames/

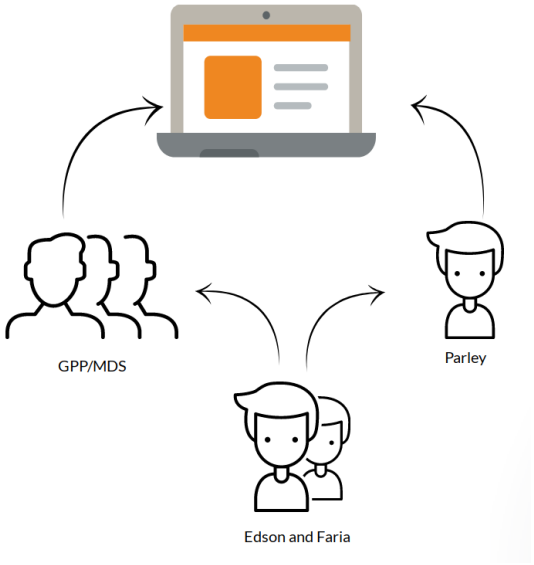

Figure 1 – Task Division

actual website with some of the features desired. The names of all the members are in the Appendix A.

Before the second half of the project, professor Edson developed the packaging template. During this term, my responsibility was to test this template in a few selected games and to evolve and maintain it, as well as to maintain and add some features to the platform developed in the previous semester. Professor Edson and Mr. Faria were code reviewers and helpers in the system.

2.3 Game Gathering

The games selected for the first part of the project were the ones developed in this department, *Faculdade UnB Gama* that is the Gama *campus* of the University since the course *Introdução aos Jogos Eletrônicos* (Introduction to Electronic Games) has been created here in the first semester of 2012. Professor Ricardo Jacobi was the first to teach the course, but it wasn't possible to contact him or get the games developed in that term. Professor Edson taught the course after that, until 2016. It has been assumed by Mr. Matheus Faria, since the beginning of this year (2017).

Because this work was being mostly held at FGA, and all the games developed here are compiled and run on Linux distributions, these were selected as first games for the platform. Another reason for this choice is the proximity with the students who created those games.

Professor Edson and Mr. Faria first contacted the students and asked them to post

their codes to GitHub. They cloned them into the `fgagamedev`[2] GitHub organization.

After that, I was responsible for checking the status of the games, gathering information such as which of them compiled, which SDL version they used, which ones had licenses. Table 2 shows these initial results.

Table 2 – Initial status of the selected games

Name	**Source?**	**License?**	**SDL**	**Compiles?**	**Year**
Deadly Wish	y	n	2	n	2016
Strife of Mythology	y	n	2	y	2016
Travelling Will	y	n	2	y	2016
7 Keys	y	MIT	2	n	2015
Babel	y	GPL 2	2	y	2015
Terracota	y	MIT	2	n	2015
Dauphine	y	n	2	n	2014
Imagina na Copa	y	n	2	y	2014
Kays Against the World	y	n	2	y	2014
Ankhnowledge	y	GPL 2	1	y	2013
The Last World War	n	-	-	-	2013
Post War	y	n	1	y	2013
War of the nets	y	GPL 2	2	y	2013
Jack the Janitor	y	GPL 3	1	y	2013
Drawing Attack	n	-	-	-	2012
Earth Attacks	n	-	-	-	2012
Emperor vs Aliens	y	n	1	y	2012
Ninja Siege	y	GPL 2	1	y	2012
Space monkeys	y	GPL 2	1	n	2012
Tacape	n	-	-	-	2012

Out of 20 games created in *Introdução aos Jogos Eletrônicos* while Professor Edson taught it, 4 didn't have a known repository and 8 didn't have a license that allowed us to change them at that time. Mr. Faria and I were responsible for finding unknown games and getting the missing licenses. As result of this task, *The Last World War* was added and 5 other had licenses acquired as shown in Table 3.

In the second semester, to test the new packaging template, four games were selected. two out of those previously chosen, developed with SDL, and two new ones developed in the first semester of 2017, made with SDL2: *Ankhnowledge*, *Ninja-Siege*, *Wenova*, and *Mindscape*, respectively. These games were chosen because they already worked correctly without any need to change their source code.

Another decision was to separate the games in a different GitHub organization, to hold all the ones developed at the University, instead of just those from FGA. Matheus

[2] https://github.com/fgagamedev/

Table 3 – Game status after contacting developers

	License	**SDL**	**Compiles**
Deadly Wish	GPL 3	2	n
Strife of Mythology	GPL 2	2	y
Travelling Will	MIT	2	y
7 Keys	MIT	2	n
Babel	GPL 2	2	y
Terracota	MIT	2	n
Dauphine	MIT	2	n
Imagina na Copa	MIT	2	y
Kays Against the World	n	2	y
Ankhnowledge	GPL 2	1	y
The Last World War	n	1	y
Post War	MIT	1	y
War of the nets	GPL 2	2	y
Jack the Janitor	GPL 3	1	y
Emperor vs Aliens	n	1	y
Ninja Siege	GPL 2	1	y
Space monkeys	GPL 2	1	n

created the `unbgames` organization for this purpose and `fgagamedev` remained as an FGA specific organization, where the packaging template is maintained, for example.

2.4 Packaging

The template for packaging was created by professor Edson is based on two main directives, modularisation and platform independence. The first one is related to dividing the directories by topic, meaning that each folder will be responsible for one thing and all the files inside of them should be related to that specific thing. The second directive, platform independence, is to make the development for multiple platforms easy. Each directory will have a division for each of the platforms.

To achieve the template modularisation, professor Edson decided to use a folder structure that would be easy to understand to anyone familiar with GNU/Linux FHS, with a few additions. Apart from the original directories in the repository, he added the folders `bin`, `dist`, `lib` and `scripts`. This structure is represented in Figure 2

- `bin` has all needed libraries and the game executable.
- `lib` All the third-party libraries should live here. The scripts to build the code are already set to look for libs inside this directory, being each subdirectory a dependency;
- `dist` This contains the files needed to generate the packages for each platform;

- `scripts` this is where the scripts to build, package and distribute the binaries for all the platforms will live. It also has a subdirectory called `utils` that holds some specific platform scripts, like generating each installer, or gather information about the host OS;

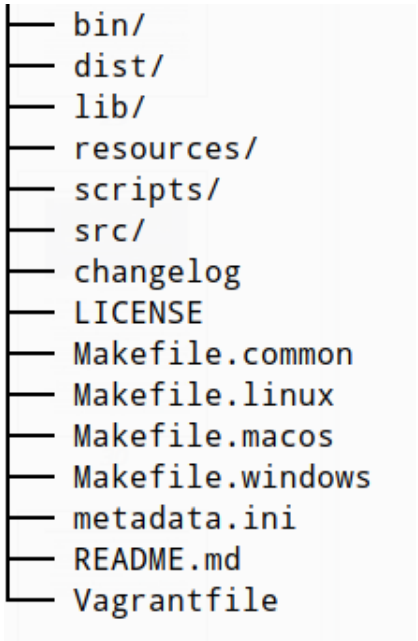

Figure 2 – Folder tree

The second directive was met by dividing some of the directories into platform limited directories, making the code that lives there accessible only when running on that individual platform. Any file outside the platform directory is considered generic and can be used for any Operating System. For example, when running on Windows, the compiler would only access generic files and Windows specific ones, like `dlls`. The same thing happens for macOS and GNU/Linux systems. This division is represented inside the `lib` directory in Figure 3.

As also seen in Figure 3, inside each platform folder (only for libraries) there is yet another division to make sure the template can generate different versions of the program for `debug` and `release`. The binaries that live on `release` are stripped of all debug symbols, resulting in smaller versions of those dependencies. Library headers go inside `include` and a compressed file with the source goes inside `src`.

The scripts kept in the `scripts` directory are the backbone of the template. Through them, it's possible to compile (creating a new executable with all the dependencies locally available), run and package a game. As long as the other files are placed correctly, the scripts work properly. There are four main and seven auxiliary scripts to accomplish these tasks, that are listed in Figure 4 and described below.

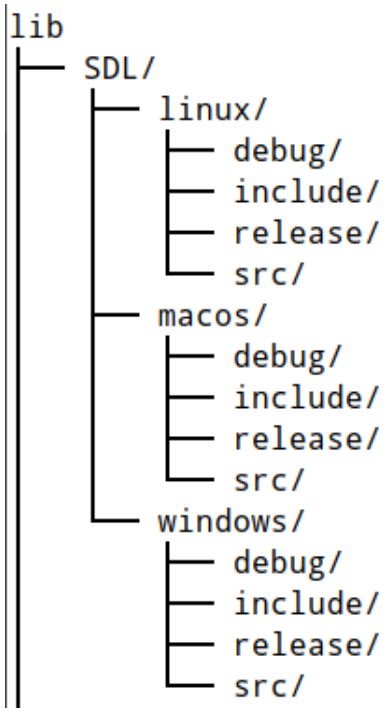

Figure 3 – Library division

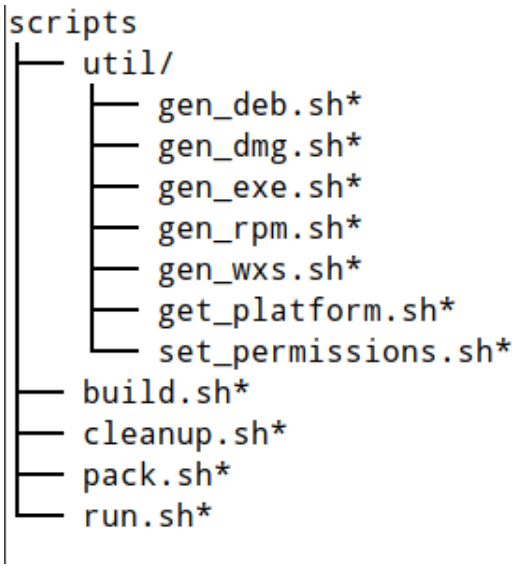

Figure 4 – Scripts

- `build.sh` builds the executable, being possible to choose which is the desired version, `debug` and `release`. Calls the appropriate `Makefile`, depending on the version and platform;
- `cleanup.sh` clears the repository, removing files generated during build and packaging, like object files and installers;
- `pack.sh` builds the release version of the program (by calling `build.sh`) and generates the installer for the specific platform it's running on. It's important to notice

that it's not possible to generate a package for a different platform from the same host system. This means that, for example, to generate Windows packages, this script must be called from within Windows and not from a Linux machine;

- `run.sh` runs the generated executable, setting the correct environment variables and pointing to where the local libs are. Attempting to run the program without this script may lead to errors;
- `util/get_platform.sh` checks and returns the current platform;
- `util/set_permissions.sh` sets files to 644[3] permission and folders to 755[4]inside a given directory;
- `util/gen_deb.sh` generates a `.deb` file to be installed in Debian-based systems;
- `util/gen_rpm.sh` generates an `.rpm` file to be installed in Red Hat based systems;
- `util/gen_exe.sh` generates the `.exe` and `.msi` to be installed on Windows systems;
- `util/gen_wxs.sh` This is called from `gen_exe` to create a `.wxs` file, that will be used to create the Windows intaller.
- `util/gen_dng.sh` creates the `.dmg` file for macOS.

All the scripts described in this section must be executed from the root folder of the repository. All paths inside the scripts are relative to that directory and running them anyplace else may cause unwanted errors.

2.5 Platform Development

The first version of the platform was developed using mixed development methods. During the first half of the semester, the Rational Unified Process and the PMBOK were used. For the next part, Scrum and XP were chosen. This choice of development framework is because of how the courses are divided.

Throughout the RUP part of the development, the team created several documents to aid the development cycle, such as vision, architecture document, class diagram, use case diagram, use case specification, test case specification.

These documents helped the team to understand the system requirements and how they should be implemented as seen in Figure 5. The most experienced members also helped the others to learn the technologies to develop the website.

[3] 644 - File owner has read and write permissions, while group and all users have only read access.
[4] 755 - File owner has read, write and execute permissions, while group and all users have read and execute permissions.

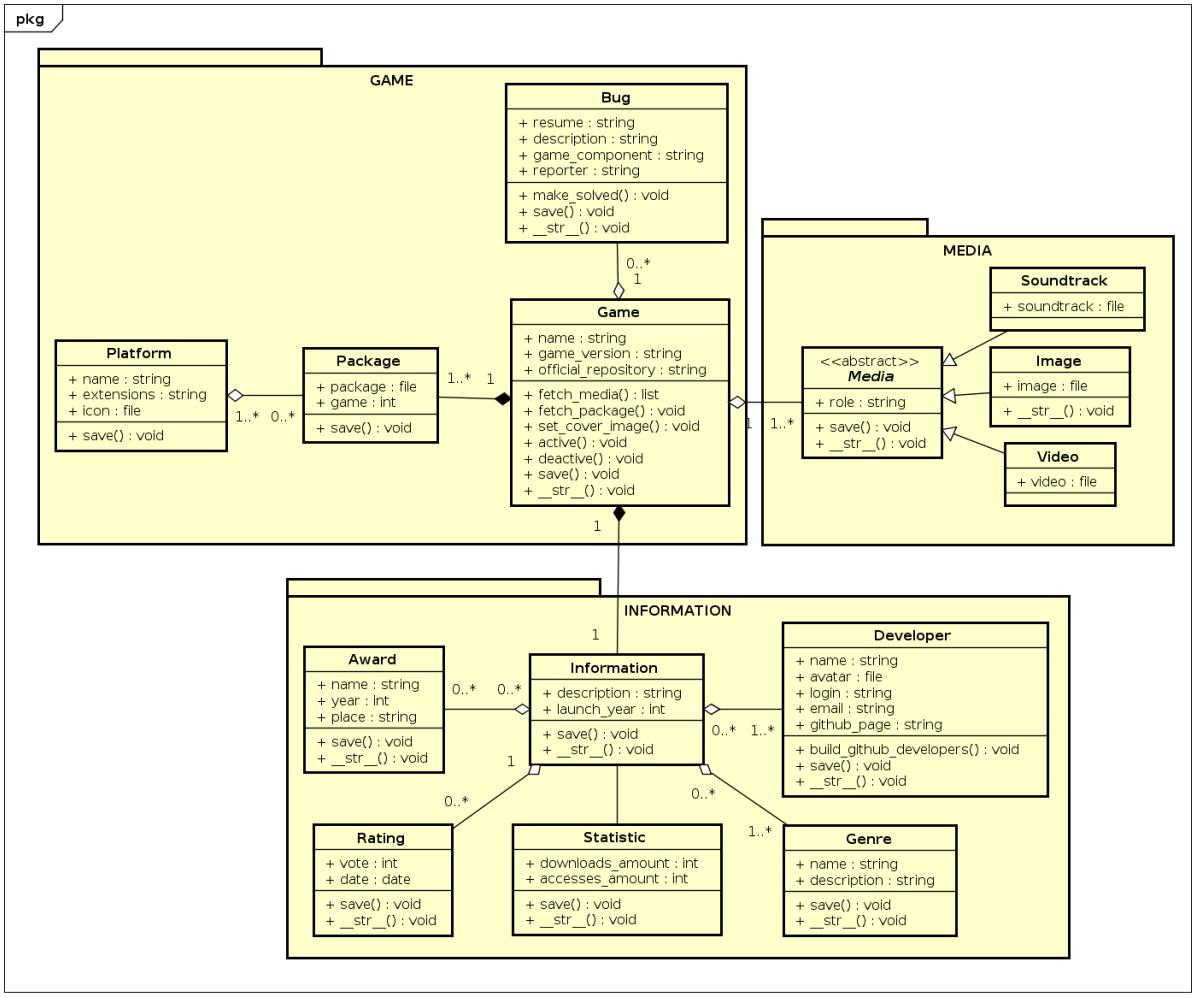

Figure 5 – Class Diagram of the Platform (UNB, 2017)

As the second part of the development started, they had to work on a totally different mindset, with new roles and documents needed. Instead of having managers, the team had now Scrum Master, Product Owner, and the Developing team (COHEN; LINDVALL; COSTA, 2003). A Scrum Master is the responsible for protecting the team, making sure knowledge is being shared and Scrum is being followed (ALLIANCE, 2017). It's important to notice that this is not equivalent to a traditional manager, that usually only bosses around the team, not caring about the people.

Product Owner is the one who will say the product value, sets the priorities and decides what need be done (AGILE42, 2017). They must assure the work meets their expectations without controlling the development team (SCHWABER; BEEDLE, 2002). The Development Team are the people who will actually do the work, they don't have a manager, they act collectively and decide how they will achieve what has to be done (GREER; HAMON, 2011).

The second half of the project focused more on the packaging template, with a

few corrections and bug reports from FGA students on the platform.

2.6 Tools

GNU Make and bash were the chosen software for building and packaging. Make is supposed to help developers managing their applications and they can run on several platforms, like Linux, Mac, and Windows. Bash is a popular script tool to manipulate files and folders from the terminal. They are distributed under GNU General Public License version 3 and the minimum required version is 4.0 (for both of them).

The chosen compilers were `gcc`, for Linux, distributed under GPL3, with at least version 5.0; `Visual Studio Compiler`, for Windows, shared with a Microsoft community License, version 2017; and `clang`, for macOS, distributed under BSD License.

For the website development, Django was picked because of the previous knowledge the group had with it. To make the front end of the application, Facebook's React was chosen for the flexibility it gives to the user interface. They are both very scalable, have a big support in the community and are released under the BSD 3-clause license. The versions being used are the last ones at the beginning of the project, namely, 1.11.1, for Django, and 15.5.4, for React.

To develop and test the template, virtual machines running Debian Jessie and CentOS 7 were used. The VMs were powered by VirtualBox 5.2, released under GPL2, that allows easy environment virtualization. It also enables a developer to test in several operating systems, which is required for the nature of this project. The computer hosting the virtual machines and used to has an Intel Core i5-6200U 2.3 GHz processor, 8 GB of RAM and an NVIDIA GeForce 940M graphic processor.

To package on Debian based systems, `lintian` version 2.5. For Red-Hat systems, `rpmlintian` version 1.9 was chosen. Both of them are distributed under GPL 2. For Windows, both Wix toolset, version 3.11, distributed under Microsoft Reciprocal License; and Gygwin shell, 2.9.0 and GPL, were used.

3 Results

This chapter explains the results obtained with the project development. Section 3.1 gives an explanation of how the model works, how it is divided and what files come when downloading it. It also explains what each file is responsible for and if the user should edit it or not. Section 3.2 shows some of the problems that came through the development/maintenance of the template and how they were overcome.

3.1 Template

One of the goals of this work was to generate a template that allowed the game developers from the courses of this University to create their games and quickly create packages to install in major Operating Systems, namely, Windows, macOS, Debian-based and Red Hat based distributions of GNU/Linux. Professor Edson made this template. I had the responsibility of testing it in a few games, evolving and maintaining it throughout all the platforms.

The template consists of a series of Bash scripts, Makefile, libraries and a directory structure that is supposed to be followed by anyone who wants to use it. It is intended to be used as a template for new games developed in the courses taught at this University, and it contains the most common libraries in game development, like SDL, SDL_image, and SDL_mixer.

3.1.1 Root directory

Currently, there are seven required files on the root directory, specifically, `LICENSE`, `Makefile.common`, `Makefile.macos`, `Makefile.windows`, `Makefile.linux`, `Vagranfile`, `changelog`, and `metadata.ini`. These files assure compilation is possible in any platform and also give some information about the project. An explanation of what each of them does and what information each may or may not have is given on Table 4. Some extra optional files are also explained.

Table 4 – Files on the root directory

File	Mode	Description
`LICENSE`	*Editable.*	This should be the text of the license or a reference to a file that has the full text. Debian packages complain if this file is the actual license text for common licenses, therefore it may be better option to only refer to a file inside the system (usually `/usr/share/common-licenses/<LICENSE>`);

`Makefile.macos`
`Makefile.windows`
`Makefile.linux`
	Uneditable.	Each of these files sets variables with specific for each system, like `CC` and `DEBUG_FLAGS`. If a variable isn't needed it will remain blank and won't change the effect of the compilation. The template is supposed to work with values as they are and users shouldn't change them unless they *actually* want a different behavior.

`Makefile.common`	*Partially editable.*	Sets some other variables, common to all OSs, like `LDFLAGS`, based on each platform Makefile. The template has set default SDL libs (SDL, SDL_image, SDL_mixer, SDL_ttf), but other external libs may be wanted. When this happens, the user should add the libs wanted to the variable `EXTERNAL_LIBS` without quotes and separated by simple space. Each of these libs must be a directory inside the `lib` folder. The rest of the file should not be changed since it may lead to major errors when using the template unless the user is sure of how it works.

Table 4 – Files on the root directory

File	Mode	Description
`Vagranfile`	*Optional.*	This file creates two Virtual Machines running Debian and CentOS. If the user wishes to give support for them both (generating both `.deb` and `.rpm` packages), they could either use the VMs or run the template natively on each system. The virtualization provides an easier way to do that, but it is up to the user deciding this detail of the development cycle.
`changelog`	*Editable.*	When creating the Debian package, it needs a changelog, that registers what was changed from the previous versions, much like a commit message. There are ways of generating this file automatically because its syntax is very particular, but the template doesn't contemplate it yet.
`metadata.ini`	*Editable.*	As the extension suggests, *ini* stands for *initialization*. It is a configuration file that follows the `ini` syntax. It defines some project properties that will be used in several steps, like building and packaging, making it a critical file to use the template correctly. The user should change this file with the appropriate information as soon as cloning the repository and throughout the development.

3.1.2 Sources Directory (`src`)

The directory that holds all source code, including headers, is called `src` and is divided in two subfolders, `engine` and `game`, as shown in Figure 6. The reason for this division is to keep separate what is engine specific (like movements, rendering windows, capturing input from the player) from the actual game. Engines can be reused in several projects, providing a basic API to create new games. Both of these directories have the same structure, that is explained in Table 5, along with the files outside them.

Table 5 – Files in the sources directory

File	Mode	Description
`main.cpp`	*Partially Editable.*	It is where the function `main` should live. This file must not be renamed or moved to inside any of the subdirectories. Users should add their logic to it, with all the relative includes. Because of compatibility issues with Windows, there is a function called `WinMain`, that only calls the main function and should not be touched.
`Makefile`	*Uneditable.*	This makefile is called during the build process, from inside `Makefile.common`. It builds the final executable, linking main with the game library, engine library, and the libraries inside `lib`.
`{game,engine}/include/*`	*Editable.*	These are the header files for the engine and the game. The template already has one header in the engine, that should not be removed, but may be renamed if the correct references are made after that. This header defines the function `resources_dir_path`, that is very important to keep the template ability to run on multiple platforms.
`{game,engine}/src`	*Editable.*	The implementation of all header functions should go inside this directory. Under this three other directories are supposed to hold platform-specific implementation, namely, `linux, windows,` and `macos`. Any code outside them is considered to be generic and can be used in any of these platforms. Every piece of code specific to one of these systems should be placed in the corresponding folder. The template already has the specific implementation to find the `resources` folder that may be renamed or reimplemented. It is not advised to change the `macos` implementation though, except for the directory name.

Table 5 – Files in the sources directory

File	Mode	Description
`{game,engine}/Makefile`	*Uneditable.*	Called from the `Makefile` in the `src` directory. Responsible for building each of these two libraries. If the folder structure was followed correctly, there is no need to change the contents of this file.

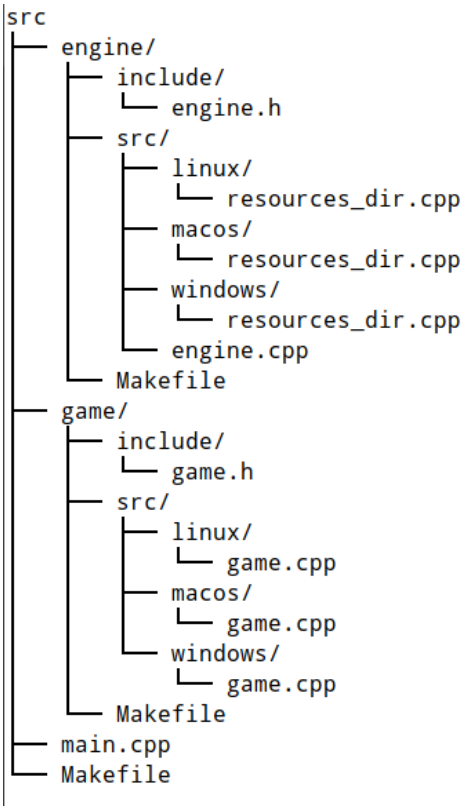

Figure 6 – `src` directory

3.1.3 Distribution folder (`dist`)

Each platform has particularities concerning generating packages. Debian, for example, requires a changelog inside the package, while Windows needs to have the package registered (with all of its contents). The dist folder contains some specific files that are needed for each package. Figure 7 shows the files needed for each system, while Table 6 explains what is of them is supposed to do.

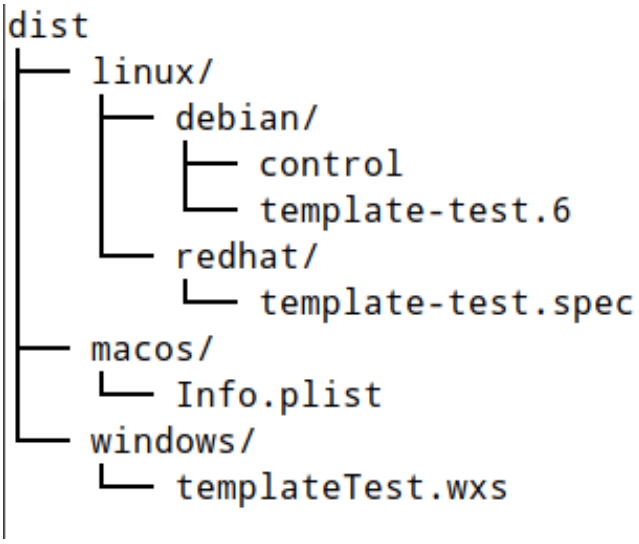

Figure 7 – `dist` directory

Table 6 – Files on the `dist` directory

File	**Mode**	**Description**
`windows/templateTest.wxs`	*Uneditable.*	This file is required to generate the installer for windows. It is an XML that lists all directories, files and libraries inside the installer. Each one of them has a unique UUID, because this is how Windows controls what is installed or removed. This file is generated once when `pack.sh` called in Windows. If the user has updated the resources and other files, they should delete this and rerun `pack.sh`, but never edit it themselves, because it is a very particular **large** file.
`macos/Info.plist`	*Uneditable.*	Because macOS packages are self-contained, this file is pretty simple. It is an XML that contains keys and values related to the package installed, like its name, version, and developer. This file has its information updated when `pack.sh` is called on a macOS system.

Table 6 – Files on the `dist` directory

File	Mode	Description
`linux/redhat/template-test.spec`	*Uneditable.*	Every `rpm` package has to have a file containing the isntructions of what to and how to install that package. This file is replaced with the specifics of each game, mostly the information in `metadata.ini`, when `pack.sh` is called on a Red Hat machine.
`linux/debian/control`	*Uneditable.*	Inside a debian package, there is a control section, that contains some metadata for the package being installed. It is a required file on every `.deb` package. This file has this data, aquired from `metadata.ini`.
`linux/debian/template-test.6`	*Uneditable.*	Even though this file is inside `debian`, it is used for both linux distributions. It is a `man` file, that contains the package usage description.

3.1.4 Scripts folder (`scripts`)

A central part of the template is the ability to build, run and package the game. This ability happens because several scripts allow users to quickly do this process, by running them from the root directory of the repository. None of these files should be modified by the user.

The scripts inside this folder are not complex or complicated since the hard work is mostly done inside the `util` directory. The build script is fairly simple, requiring one argument that is the mode the script will run, `debug` or `release`. If none is provided, it will use `debug` as default. It simply checks the platform and run the command `make` with the appropriate Makefile and mode; `run.sh` sets some variables and change the directory to where all the libs are to then call the executable; `cleanup.sh` remove objects, libraries and other files generated during compilation; and `pack.sh` calls one of the scripts inside `util` to generate the corresponding package.

Generating a `.deb` package consists in a few steps as shown in Listing 1. It first sets some variables and loads info from `metadata.ini`, in lines 7-11. When the function `gen_deb()` is called, it creates a temporary directory and its structure, in lines 9 through

24. From line 26 through 43, the executable, the required libs, and the resources are all copied to their respective location inside this structure. The `control` file is copied from `dist` folder and the information is replaced with what is in `metadata.ini` in lines 46-59. Lines 62-75 create some other directories, copy license, changelog and man pages to a documentation folder, and compress some of the files. Lines 77-88 set the correct permissions, strip the executable, builds and checks the package for errors.

Listing 1 – `gen_deb.sh`

```
1  #!/bin/bash
2  #
3  # Generates .deb package for Linux
4  #
5
6  # Include project metadata
7  . metadata.ini
8
9  PACKAGE_NAME=$EXECUTABLE_NAME
10 PACKAGE_VERSION=$VERSION_MAJOR.$VERSION_MINOR-$VERSION_RELEASE
11 OUTPUT_FILE=$PACKAGE_NAME\_$PACKAGE_VERSION.deb
12
13 function gen_deb()
14 {
15     # Build dir
16     tmp_dir=/tmp/$PACKAGE_NAME\_$PACKAGE_VERSION
17     rm -rf $tmp_dir
18     mkdir -p $tmp_dir
19
20     # Data dir: resources, scripts and executable
21     var_dir=$tmp_dir/var
22     data_dir=$var_dir/games
23     install_dir=$data_dir/$PACKAGE_NAME
24     mkdir -p $install_dir
25
26     cp src/$EXECUTABLE_NAME\_release $install_dir/$EXECUTABLE_NAME
27
28     lib_dir=$install_dir/lib
29     mkdir -p $lib_dir
30
31     for extlib in `ls lib`;
32     do
33         cp -P lib/$extlib/linux/release/* $lib_dir;
34     done
35
36     # Removing embedded libraries
37     rm $lib_dir/libjpeg*
38     rm $lib_dir/libpng*
```

```
39
40      resources_dir=$install_dir/resources
41      mkdir -p $resources_dir
42
43      cp -r resources/* $resources_dir/
44
45      # Launcher script dir
46      usr_dir=$tmp_dir/usr
47      exec_dir=$usr_dir/games
48      mkdir -p $exec_dir
49
50      printf "#!/bin/bash\nexport LD_LIBRARY_PATH=/var/games/$PACKAGE_NAME
        /lib && cd /var/games/$PACKAGE_NAME/ && ./$EXECUTABLE_NAME" >
        $exec_dir/$EXECUTABLE_NAME
51
52      # Debian package info dir
53      mkdir -p $tmp_dir/DEBIAN
54      cp dist/linux/debian/control $tmp_dir/DEBIAN/
55      sed -i -- 's/%%PACKAGE_NAME%%/'"$PACKAGE_NAME"'/' $tmp_dir/DEBIAN/
        control
56      sed -i -- 's/%%PACKAGE_VERSION%%/'"$PACKAGE_VERSION"'/' $tmp_dir/
        DEBIAN/control
57      sed -i -- 's/%%MAINTAINER_NAME%%/'"$MAINTAINER_NAME"'/' $tmp_dir/
        DEBIAN/control
58      sed -i -- 's/%%MAINTAINER_CONTACT%%/'"$MAINTAINER_CONTACT"'/'
        $tmp_dir/DEBIAN/control
59      sed -i -- 's/%%GAME_DESCRIPTION%%/'"$GAME_DESCRIPTION"'/' $tmp_dir/
        DEBIAN/control
60
61      # Documentation
62      share_dir=$tmp_dir/usr/share
63      doc_dir=$tmp_dir/usr/share/doc/$PACKAGE_NAME
64      mkdir -p $doc_dir
65
66      cp changelog $doc_dir/changelog.Debian
67      cp LICENSE $doc_dir/copyright
68      gzip -n9 $doc_dir/changelog.Debian
69
70      man_dir=$share_dir/man
71      section_dir=$man_dir/man6
72      mkdir -p $section_dir
73
74      cp dist/linux/debian/template-test.6 $section_dir/$PACKAGE_NAME.6
75      gzip -n9 $section_dir/$PACKAGE_NAME.6
76
77      # Set the permissions
78      scripts/util/set_permissions.sh $tmp_dir
```

```
79      chmod 755 $exec_dir/$EXECUTABLE_NAME
80      chmod 755 $install_dir/$EXECUTABLE_NAME
81
82      # Strip executable debug symbols
83      strip $install_dir/$EXECUTABLE_NAME
84
85      # Build and check the package
86      fakeroot dpkg-deb --build $tmp_dir
87      mv /tmp/$OUTPUT_FILE .
88      lintian $OUTPUT_FILE
89  }
90
91  echo "Generating "$OUTPUT_FILE "..."
92  gen_deb
93  echo "Done"
```

Making an `.rpm` package is somewhat simpler than generating a Debian package. As seen in Listing 2, `gen_rpm.sh` starts on lines 7-11 also loading `metadata.ini` and defining some variables. When `gen_rpm()` is called, it first calls the rpm tool that creates a folder structure for the package, on line 18. After that, on lines 21 - 29, it copies the spec file to its place on that structure and replaces the information read from `metadata.ini`. Line 32 puts the text script that will be executed in the folder structure. Lines 35-39 remove any traces of other times this script was called and create a `tar` package based on the structure the rpm builder created. Lines 42-44 build the `rpm` package and calls the lint to check it. Unlike Debian, everything the builder needs to know is inside the `spec` file; the script only copies things to where they are supposed to be.

Listing 2 – `gen_rpm.sh`

```
1  #!/bin/bash
2  #
3  # Generates .deb package for Linux
4  #
5
6  # Include project metadata
7  . metadata.ini
8
9  PACKAGE_NAME=$EXECUTABLE_NAME
10 PACKAGE_VERSION=$VERSION_MAJOR.$VERSION_MINOR-$VERSION_RELEASE
11 OUTPUT_FILE=$PACKAGE_NAME\_$PACKAGE_VERSION.rpm
12
13 function gen_rpm()
14 {
15     work_dir=‘pwd‘
16
17     # RPM build dir setup
```

```
18      rpmdev-setuptree
19
20      # Preparing the spec file
21      spec_file=$PACKAGE_NAME.spec
22      cp dist/linux/redhat/template-test.spec ~/rpmbuild/SPECS/$spec_file
23      cp dist/linux/debian/template-test.6 dist/linux/debian/$PACKAGE_NAME
        .6
24
25      sed -i -- 's/%%PACKAGE_NAME%%/'$PACKAGE_NAME'/g' ~/rpmbuild/SPECS/
        $spec_file
26      sed -i -- 's/%%VERSION_MAJOR%%/'$VERSION_MAJOR'/g' ~/rpmbuild/SPECS/
        $spec_file
27      sed -i -- 's/%%VERSION_MINOR%%/'$VERSION_MINOR'/g' ~/rpmbuild/SPECS/
        $spec_file
28      sed -i -- 's/%%VERSION_RELEASE%%/'$VERSION_RELEASE'/g' ~/rpmbuild/
        SPECS/$spec_file
29      sed -i -- 's/%%GAME_DESCRIPTION%%/'"$GAME_DESCRIPTION"'/g' ~/
        rpmbuild/SPECS/$spec_file
30
31      # Launcher script dir
32      printf "#!/bin/bash\nexport LD_LIBRARY_PATH=/var/games/$PACKAGE_NAME
        /lib && cd /var/games/$PACKAGE_NAME/ && ./$EXECUTABLE_NAME\n" > dist/
        linux/redhat/$EXECUTABLE_NAME
33
34      # Preparing the source package
35      rm -rf /tmp/$PACKAGE_NAME-$VERSION_MAJOR.$VERSION_MINOR
36      mkdir -p /tmp/$PACKAGE_NAME-$VERSION_MAJOR.$VERSION_MINOR
37      cp -r * /tmp/$PACKAGE_NAME-$VERSION_MAJOR.$VERSION_MINOR/
38      cd /tmp && tar -czpf ${PACKAGE_NAME}.tar.gz $PACKAGE_NAME-
        $VERSION_MAJOR.$VERSION_MINOR/
39      cp /tmp/${PACKAGE_NAME}.tar.gz ~/rpmbuild/SOURCES/
40
41      # Build and check the package
42      cd ~/rpmbuild/SPECS && rpmbuild -ba $spec_file
43      cp ~/rpmbuild/RPMS/x86_64/* $work_dir
44      cd $work_dir && rpmlint $PACKAGE_NAME-$VERSION_MAJOR.*
45  }
46
47  echo "Generating "$OUTPUT_FILE "..."
48  gen_rpm
49  echo "Done"
```

To generate the Windows installer, the script `gen_exe.sh` is called. It starts, in lines 7-11, loading and setting variables just like the other scripts. When `gen_exe()` is called, it makes the folder where all the libs and resources will live, in lines 14-15. Lines 17-20 copy the release libs to the folder created in the previous step. Lines 22-25 create

a new `wxs` file, only if one doesn't exist. This decision was made because it takes a lot of time to create that file. If the user wants to recreate it, they just have to delete it from the `dist` folder. Lines 27-36, copy all the libs, resources, executable and wxs to a temporary directory. Lines 38-41 do the actual package building, calling `candle.exe` and `light.exe` that are Wix tools that compile the `.wxs` file into `.wxsobj` and create the `.msi`, respectively.

Listing 3 – `gen_exe.sh`

```
1  #!/bin/bash
2  #
3  # Generates .exe installer for Windows
4  #
5
6  # Include project metadata
7  . metadata.ini
8
9  WXS_PATH="dist/windows/$PACKAGE_NAME.wxs"
10 OUTPUT_FILE=$EXECUTABLE_NAME.exe
11 PACKAGE_VERSION=$VERSION_MAJOR.$VERSION_MINOR.$VERSION_RELEASE
12
13 function gen_exe() {
14   rm -rf bin/windows
15   mkdir -p bin/windows
16
17   for DIR in $(ls -D lib);
18   do
19     cp -P lib/$DIR/windows/release/* bin/windows
20   done;
21
22   if ! [ -e $WXS_PATH ];
23   then
24     scripts/util/gen_wxs.sh
25   fi
26
27   mkdir -p .tmp
28   cp -u src/$EXECUTABLE_NAME\_release .tmp/$OUTPUT_FILE
29
30   cp -u bin/windows/* .tmp/
31   cp -f $WXS_PATH .tmp/$PACKAGE_NAME.wxs
32
33   # cp -u dist/windows/Manual.pdf .tmp/
34   cp -ur resources .tmp/
35
36   cd .tmp
37
38   candle.exe $PACKAGE_NAME.wxs
```

```
39    light.exe -sice:ICE60 -ext WixUIExtension $PACKAGE_NAME.wixobj
40    cp $PACKAGE_NAME.msi ..
41    cd ..
42  }
43
44  echo "Generating "$OUTPUT_FILE "..."
45  gen_exe
46  echo "Done"
```

3.2 Difficulties

Creating the installer for Windows has proved to be the hardest part of the template because Windows has an entirely different folder structure from GNU/Linux systems, they also don't have the same tools available (like Bash). Compiling for Windows has also turned out to be more challenging than Professor Edson first anticipated, because the template wouldn't run correctly, even after installing all required dependencies.

The template for Windows was supposed to use Visual Studio compiler, which is a tool made specifically for that platform, however when calling the compiler, it would not find any of the `.cpp` files. To try to revert that situation, the parameters passed to the compiler inside `Makefiles` were checked and the compiling commands were run individually inside each folder that had the source code. Even after that thorough examination, the compiler would refuse to find the files. All tools were uninstalled and reinstalled, and the problem remained. It was decided to change the compiler to `gcc` to solve this issue, just like the GNU/Linux systems.

Changing the compiler was partially natural because it was needed only to replicate the Linux `Makefile` on Windows (with a few commands replaced). It was required to install one more dependency, the compiler, and its stack, but Visual Studio could be dropped too. This error has caused another complication, because, for some reason, during the final part of the compilation, it didn't recognize the `main` function. It turns out that the compiler needed a different entry point instead of the default `main`. According to Visual Studio documentation, when creating a GUI application, it requires a function called `WinMain` (MICROSOFT, 2017a) and even with `mingw` it complained about not having it. This function was added, and it simply called `main`.

After compiling, the issue was to generate the installer. Initially, the script didn't provide any means to create the required `wxs` with the data from the repository, demanding the user to create that file manually. It wasn't an easy task, since this is a very particular large file, with specific tags, keywords, and syntax. For example, every independent set of data must be wrapped around a component, (that is how Windows checks what is what inside a package) and each of the resources inside the package must be

listed).

To aid in that process, the script `gen_wxs` was created and the `wxs` generation was divided into three parts, header, directory, and feature, just to make it easier to generate the whole file. The main problem in this part of the template development was finding and listing the resources because there could be any number of subdirectories. Recursion was the first idea to solve this issue, as seen lines 189-207 of Listing 4, but it proved to be hard to use in Bash because it defines variables only once. The recursive variable had to be updated, to solve that issue, before returning to the previous call. The command in line 206 of Listing 4 removes everything in the name of the file until the last `/`, assuring that `$FILE_PATH` has the correct value on the next recursive call. By doing that, it was possible to list all resources and their folders in the `wxs` file.

Listing 4 – Part of `gen_wxs.sh`

```
189 function check_directory_for_file() {
190   BASE_DIR=$1
191
192   for FILE in $(ls $BASE_DIR);
193   do
194     FILE_PATH="$BASE_DIR/$FILE"
195     if [ -d $FILE_PATH ];
196     then
197       append_directory_tag $FILE_PATH
198       check_directory_for_file $FILE_PATH
199       close_tag "Directory"
200     else
201       append_component_tag $FILE_PATH
202       append_file_tag $FILE_PATH
203       close_tag "Component"
204     fi
205   done
206   BASE_DIR=${BASE_DIR%/*}
207 }
```

Another challenge in testing the template was the migration to SDL2. Even though it was intended to be used with SDL2 since the beginning, Professor Edson chose to start with SDL and then add support to the newer version. The games initially selected for this part were *Traveling Will* and *Deadly Wish*, from the beginning of 2015. Even though they work fine when the libs are installed, there was a problem using them to test the template, because they needed the external engine created for the course a few years ago. This engine was built to be used as a shared library, which is fine and encouraged, but it expected a different folder structure then the template offered. Even after a few minor changes in it, the game still didn't adequately run when packaged. These errors might be happening from the version of the engine being used because the games didn't

specify which they needed. Since the goal of this work was to test the template and not to fix/maintain the engine or the games, *Traveling* and *Deadly* were dropped. The new games selected were *Wenova* and *Mindscape*, both developed on the first semester of 2017.

Changing the template to support SDL2 was a little tricky. Different than the previous SDL version, on Linux the libraries didn't work on both of the desired distributions out of the box, requiring specific binaries for Debian based systems and Red-Hat based systems. Still on Linux, playing `.mp3` files proved to be slightly more complicated in SDL2. To read that extension SDL_mixer needs to be compiled with the `smpeg` library, that will be loaded as a shared lib with the program. Even with the library installed, SDL_mixer would refuse to open `.mp3` songs, saying it wasn't a recognized format. The solution came after carefully observing the output of the `configure` script, that checks which dependencies and third party libs are installed in the computer building SDL. It turns out that SDL2_mixer required version 2 of `smpeg`, but that wasn't discriminated anywhere in their documentation. SDL_mixer 1, used version 0.4.5 of `smpeg`, the one that comes in the distro repositories, and worked fine because of that.

On Windows, the problem was loading the images of *Wenova*. Since `mingw` is being used as compiler for Windows, the `mingw` binaries were downloaded. It turns out that, for some reason, SDL_image would not load `.png` files, due to `libpng` having some reference to some function that wasn't defined in any part of it or its dependencies. The error on the console wasn't of any help, and the internet searches would only result in adding `zlib1` to the dependencies, which was already done. In an attempt to correct the error, just to try something, I unzipped the Visual Studio binaries, which in theory wouldn't run on `mingw` and tried recompiling the game. For some unknown reason, probably all the dependencies were corectly put inside this version of `libpng`, the game ran correctly this time.

Mindscape was probably the hardest game to compile in both platforms, because of a silly mistake the developers made, that was masked in the source code. For some reason, maybe incorrectly merging developing branches or change of the initial plans for the game, they had defined two headers called `game.hpp`, one in the engine, the other in the game. When creating a C/C++ header, it's a good practice to guard it against multiple imports using a `#IF_N_DEF` macro that will prevent users to import the same header accidentally (redefining it) (DISCH, 2009). The team that made `Mindscape` naturally used this guarding against multiple imports, but they hard referenced all of their imports, therefore always importing the engine `game.hpp`, that had all the definitions they needed. The template works differently, passing the path of the headers to the compiler, instead of putting that address hardcoded in the source code. Because of that, when removing the full path in the `.cpp` files, the compiler would find the first defined `game.hpp`, which was in the game folder. This file had no definitions other the constructor and the destructor, causing a lot of `Undefined reference` errors during compilation. It took a long time to

find out, and it was almost an accident, that there were two of these files, one empty and the other with all the functions correctly defined. Once this was solved, the game compiled and ran successfully.

4 Conclusion

Creating games is something supposed to be challenging and fun. But just like other types of software, it requires a lot of effort to distribute them to the player, mainly if a developer targets multiple platforms and Operating Sytems. Each machine has a different configuration, different libraries, and architecture and most of the development inside the University doesn't take that into account, which may cause the final software to have unexpected errors.

This work has shown just how difficult it is to support multiple OSs, seeking to create a unique solution to easier this task. Even with the care of separating the specifics for each Operating System and package the binaries within each, this project hasn't worked the way it was expected to. It wasn't possible to test the macOS distribution, due to lack of time, and Windows just gave a lot of other problems, like issues with the local environment, executables that worked partially, runtime errors (that didn't happen on Linux, for example).

With all these problems, the project served best as a learning experience, from which everyone involved should take a few lessons. The first lesson is that creating the template the way it was made wasn't the correct approach because all of the parties tried to replicate the macOS "way of packaging" on *all* the platforms, instead of focusing on how to follow each platform "rules." Using a self-contained package might have seemed a good idea, but that's not how Windows or Linux work.

Another lesson is that we should take advantage of the natural environment in each platform, by creating files to work with Visual Studio and XCode instead of 'forcing' the way around with Bash and Make. On Windows, even with the initial idea of using Visual Studio compiler, we still had Bash scripts, and when VS didn't work, we used another Linux solution, which may be the cause of all the errors and issues that happened on that OS.

The third lesson is concerning the libraries used in the project. To make all the libs available for the player it was decided to use binaries and not the source files in the template (even though the source is in it too, but just to use in extreme cases). Every binary carries some information of how it was built, and that may cause problems if the computer running them doesn't have all the dependencies that were there at compiling time. In the future, the source probably will be used.

On top of that, there is also the development and maintaining of the *UnB Games* website. Maintaining a free software project demands time and volunteers that want to work with that software, but it also needs a well-written documentation, to aid people that

will contribute to it, and guidelines to prevent the mess of everyone coding the way they want to. The website is still at a very early stage on that matter, with poor documentation and without guidelines for someone who wants to help, but this will change soon.

The full project has shown that game developing in this University is much better than previous years, but it still has a long way to go. The template must be improved to facilitate the distribution of these games to society. The platform has to mature to receive contributions from other people.

Bibliography

AGILE42. *Scrum Roles*. 2017. Disponível em: <http://www.agile42.com/en/agile-info-center/scrum-roles/>. Cited on page 29.

ALLBERY, B. S. et al. Filesystem hierarchy standard. 2015. Cited 3 times on pages 9, 18, and 19.

ALLIANCE, S. *Scrum Roles Desmystified*. 2017. Disponível em: <https://www.scrumalliance.org/agile-resources/scrum-roles-demystified>. Cited on page 29.

BANDEL, D.; NAPIER, R. *Special Edition Using Linux*. Que, 2001. (Special Edition Using Series). ISBN 9780789725431. Disponível em: <https://books.google.com.br/books?id=_HEhAQAAIAAJ>. Cited on page 18.

BETHKE, E. *Game Development and Production*. Wordware Pub., 2003. (Wordware game developer's library). ISBN 9781556229510. Disponível em: <https://books.google.com.br/books?id=m5exIODbtqkC>. Cited on page 16.

BUNDLE, H. *What is Featured Charity*. 2017. Disponível em: <https://support.humblebundle.com/hc/en-us/articles/115009679508-What-is-Featured-Charity->. Cited on page 21.

CAMPBELL, J. G. Algorithms and data structures for games programming. 2009. Cited on page 17.

CHACON, S.; STRAUB, B. *Pro git*. [S.l.]: Apress, 2014. Cited on page 18.

COHEN, D.; LINDVALL, M.; COSTA, P. Agile software development. *DACS SOAR Report*, v. 11, 2003. Cited on page 29.

CRAWFORD, C. *The Art of Computer Game Design*. Berkeley, CA, USA: Osborne/McGraw-Hill, 1984. ISBN 0881341177. Cited on page 16.

DISCH. *Headers and Includes: Why and How*. 2009. Disponível em: <http://www.cplusplus.com/forum/articles/10627/>. Cited on page 45.

FISHER, T. *What is the Windows Registry and What's it Used For?* 2017. Disponível em: <https://www.lifewire.com/windows-registry-2625992>. Cited on page 20.

GOG. *GOG Galaxy*. 2017. Disponível em: <https://www.gog.com/galaxy>. Cited on page 21.

GORDON, R. 2017. Disponível em: <https://plus.google.com/+RyanGordon/posts/TB8UfnDYu4U>. Cited on page 17.

GREER, D.; HAMON, Y. Agile software development. *Software: Practice and Experience*, Wiley Online Library, v. 41, n. 9, p. 943–944, 2011. Cited on page 29.

LINODE. *Linux Package Management*. 2017. Disponível em: <https://www.linode.com/docs/tools-reference/linux-package-management>. Cited on page 19.

LOELIGER, J.; MCCULLOUGH, M. *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*. O'Reilly Media, Incorporated, 2012. (Oreilly and Associate Series). ISBN 9781449316389. Disponível em: <https://books.google.com.br/books?id=ZkXELyQWf4UC>. Cited on page 18.

MICROSOFT. */ENTRY (Entry-Point Symbol)*. 2017. Disponível em: <https://docs.microsoft.com/en-us/cpp/build/reference/entry-entry-point-symbol>. Cited on page 43.

MICROSOFT. *Registry*. 2017. Disponível em: <https://msdn.microsoft.com/en-us/library/windows/desktop/ms724871(v=vs.85).aspx>. Cited on page 20.

MICROSOFT. *Structure of the Registry*. 2017. Disponível em: <https://msdn.microsoft.com/en-us/library/windows/desktop/ms724946(v=vs.85).aspx>. Cited on page 20.

MICROSOFT. *Windows registry information for advanced users*. 2017. Disponível em: <https://support.microsoft.com/en-us/help/256986/windows-registry-information-for-advanced-users>. Cited on page 20.

MITCHELL, S. *SDL Game Development*. Packt Publishing, 2013. (Community experience distilled). ISBN 9781849696838. Disponível em: <https://books.google.com.br/books?id=SbmfrHIlhK4C>. Cited on page 17.

PRESSMAN, R. *Software Engineering: A Practitioner's Approach*. 7. ed. New York, NY, USA: McGraw-Hill, Inc., 2010. ISBN 0073375977, 9780073375977. Cited on page 16.

SCHWABER, K.; BEEDLE, M. *Agile software development with Scrum*. [S.l.]: Prentice Hall Upper Saddle River, 2002. v. 1. Cited on page 29.

SDL. *Introduction to SDL 2.0*. 2017. Disponível em: <https://wiki.libsdl.org/Introduction>. Cited on page 17.

SPLITPLAY. *O que estamos construindo?* 2017. Disponível em: <http://splitplay.strikingly.com/>. Cited on page 21.

STEAM. *Welcom to Steam*. 2017. Disponível em: <http://store.steampowered.com/about/>. Cited on page 20.

TEAM, B. D. *The /usr Versus /usr/local Debate*. 2017. Disponível em: <http://www.linuxfromscratch.org/blfs/view/svn/introduction/position.html>. Cited on page 19.

UNB, P. de J. *Documento de Arquitetura*. 2017. Disponível em: <https://github.com/fga-gpp-mds/2017.1-PlataformaJogosUnB/wiki/Documento-de-Arquitetura>. Cited 2 times on pages 8 and 29.

WEBSTER, M. *Definition of repository*. 2017. Disponível em: <https://www.merriam-webster.com/dictionary/repository>. Cited on page 17.

WESTBY, E. *Git for Teams: A User-Centered Approach to Creating Efficient Workflows in Git*. O'Reilly Media, 2015. ISBN 9781491911211. Disponível em: <https://books.google.com.br/books?id=73FrCgAAQBAJ>. Cited on page 18.

WIKIPEDIA. *File System*. 2017. Disponível em: <https://en.wikipedia.org/wiki/File_system>. Cited on page 20.

WIKIPEDIA. *GOG.com*. 2017. Disponível em: <https://en.wikipedia.org/wiki/GOG.com>. Cited on page 21.

WIKIPEDIA. *Steam*. 2017. Disponível em: <https://en.wikipedia.org/wiki/Steam_(software)>. Cited on page 20.

Appendix

APPENDIX A – Members of GPP/MDS team

The following students were the direct responsible for developing the first version of the platform. They are students of the courses *Métodos de Desenvolvimento de Software* and *Gestão de Portfolios e Projetos* ministered by Professor Carla Silva Rocha Aguiar.

- Arthur Temporim
- Artur Bersan
- Eduardo Nunes
- Ícaro Pires de Souza Aragão
- João Robson
- Letícia de Souza
- Marcelo Ferreira
- Matheus Miranda
- Rafael Bragança
- Thiago Ribeiro Pereira
- Varley Santana Silva
- Victor Leite
- Vinicius Ferreira Bernardo de Lima

APPENDIX B – Selected games

This appendix shows the authors, year of publication, quantity of players, genre and description, whenever possible, of each selected game for this first part of the project.

B.1 Jack the Janitor

Authors: Athos Ribeiro, Alexandre Barbosa, Mateus Furquim, Átilla Gallio

Year: 1/2013

Genre: Puzzle, platform

 # Players: Single player

Repository: <https://github.com/fgagamedev/Jack-the-Janitor>

Description[1]**:** Jack, The Janitor is a puzzle game where the player controls Jack, a school's janitor who must organize the school's warehouse. Jack can push boxes to the left or to the right and jump boxes.

When Jack fills an entire row with boxes, they disappear from the screen and go to a small window on the right side of he screen called the closet.

The closet shows how Jack organized the rows of boxes. When similar boxes are combined in the closet, Jack gets extra points and some power ups (to be implemented).

The game ends if a falling box hits Jack or if the closet gets full.

B.2 Emperor vs Aliens

Authors: Leonn Ferreira, Luis Gustavo

Year: 2/2012

Genre: Tower defense

 # Players: Single player

Repository: <https://github.com/fgagamedev/Emperor-vs-Aliens>

[1] Available on the repository README.md

B.3 Ninja Siege

Authors: Tiago Gomes Pereira, Matheus Fonseca, Charles Oliveira, Pedro Zanini

Year: 2/2012

Genre: Tower defense

 # Players: Single player

Repository: <https://github.com/fgagamedev/Ninja-Siege>

Description: The ninja academy is being raided and you have to defend it.

B.4 Space Monkeys

Authors: Victor Cotrim

Year: 2/2012

Genre: Tower defense

 # Players: Single player

Repository: <https://github.com/fgagamedev/Space-Monkeys>

Description: Monkeys are attacking your home planet. They come in waves and you have to get rid of them all.

Remarks: It's interest to notice that, by this time, the students of *Introdução aos Jogos Eletrônicos* didn't have designers with them in the team. Figure 8 shows that, given the complexity of developing a game, sometimes the artwork was not a priority. This is also one of the games that didn't run properly after the compilation.

B.5 War of the Nets

Authors: Matheus Faira, Lucas Kanashiro, Luciano Prestes, Lucas Moura

Year: 2/2013

Genre: Turn Based Strategy

 # Players: Multiplayer on LAN

Repository: <https://github.com/fgagamedev/War-of-the-Nets>

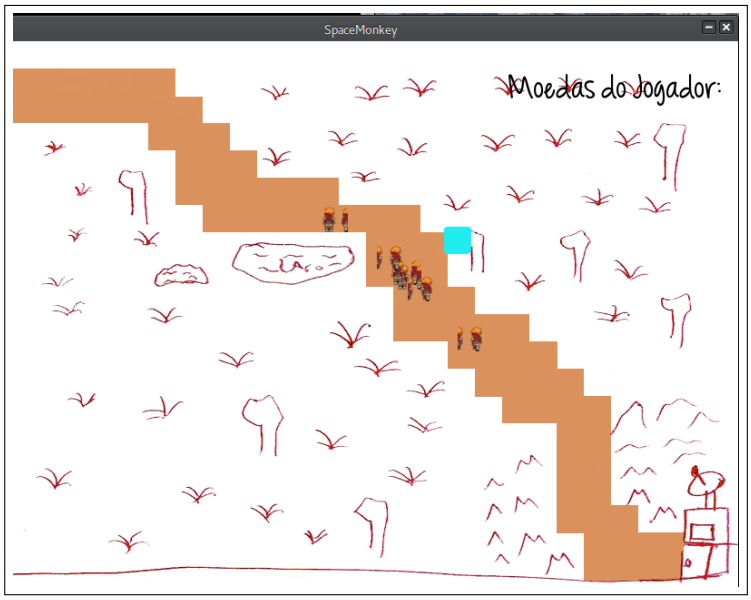

Figure 8 – Space Monkey

Description: It is a turn based strategy (TBS), where the objective is to construct a network from the base to a right point, faster than your enemy. You also can destroy his network with bombs, or infiltrate it with spies.

B.6 Post War

Authors: Bruno de Andrade, Jonathan Rufino, Yago Regis

Year: 2/2013

Genre: Turn Based Strategy

 # Players: Multiplayer on LAN

Repository: <https://github.com/fgagamedev/Post-War>

B.7 Ankhnowledge

Authors: Arthur del Esposte, Alex Campelo, Atilla Gallio

Year: 2/2013

Genre: Turn Based Strategy

 # Players: Multiplayer on LAN

Repository: <https://github.com/fgagamedev/Ankhnowledge>

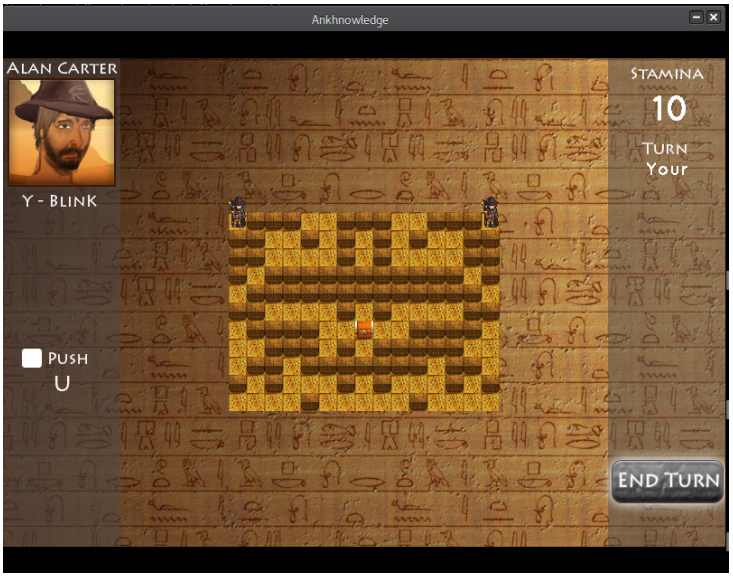

Figure 9 – Ankhnowledge

Remarks: From the games developed before the time the course was taught in conjunction with the students from *Darcy Ribeiro*, this is one of the prettiest and most pleasant games to play. Because one of the students is a software developer and designer, the user interface was very well drawn as seen in Figure 9.

B.8 Last World War

Authors: Gabriela Navarro

Year: 2/2013

Genre: Turn Based Strategy

 # Players: Multiplayer on LAN

Repository: <https://github.com/fgagamedev/LastWorlWar>

B.9 Kays against the World

Authors: Carlos Coelho, Bruno de Amorim Campos, Bruno Carbonell, Guilherme Fenterseifer, Fernando Tollendal, Lucas Sanginez, Victor Bednarczuk

Year: 1/2014

Genre: Platform

 # Players:

Repository: <https://github.com/fgagamedev/Kays-Against-the-World>

B.10 Imagina na Copa

Authors: Iago Mendes Leite, Jonathan Henrique Maia de Moraes, Luciano Henrique Nunes de Almeida, Inara Régia Cardoso, Renata Rinaldi, Lucian Lorens Ramos

Year: 1/2014

Genre: Platform

 # Players: Single player

Repository: <https://github.com/fgagamedev/Imagina-na-Copa>

B.11 Dauphine

Authors: Caio Nardelli, Simiao Carvalho

Year: 1/2014

Genre: Platform

 # Players: Single player

Description: A platforming/stealth game in a medieval fantasy setting, developed with SDL2.

Repository: <https://github.com/fgagamedev/Dauphine>

B.12 Terracota

Authors: Álvaro Fernando, Macartur Sousa, Carlos Oliveira, André Coelho, Pedro Braga, Wendy Abreu, José de Abreu

Year: 1/2015

Genre: Adventure

 # Players: Single player

Repository: <https://github.com/fgagamedev/Terracota>

B.13 7 Keys

Authors: Paulo Markes, Bruno Contessotto Bragança Pinheiro, Lucas Rufino, Luis André Leal de Holanda Cavalcanti, Maria Cristina Monteiro de Oliveira, Guilherme Henrique Nunes Lopes

Year: 1/2015

Genre: Adventure

 # Players: Single player

Repository: <https://github.com/fgagamedev/7-Keys>

B.14 Babel

Authors: Álex Silva Mesquita, Jefferson Nunes de Sousa Xavier, Rodrigo Gonçalves, Vinícius Corrêa de Almeida, Heitor Campos, Max Von Behr, Aleph Telles de Andrade Casara, Washington Rayk

Year: 1/2015

Genre: Adventure

 # Players: Single player

Repository: <https://github.com/fgagamedev/Babel>

Description: The mankind wanders the universe looking for a new habitable planet. They found an unknown planet with a big and strange tower.

The challenge is explore the tower and the planet and expand your resources, but be careful with the mysteries of this new planet.

B.15 Strife of Mithology

Authors: Jônnatas Lennon Lima Costa, Marcelo Martins de Oliveira, Victor Henrique Magalhães Fernandes, Dylan Jefferson M. Guimarães Guedes

Year: 1/2016

Genre: Tower Defense

 # Players: Single player

Repository: <https://github.com/fgagamedev/Strife-of-Mithology>

Description: A 2d-isometric Tower Defense based on mythology.

B.16 Traveling Will

Authors: João Araújo, Vitor Araujo, Igor Ribeiro Duarte, João Paulo Busche da Cruz

Year: 1/2016

Genre: Platform, Runner

 # Players: Single player

Repository: <https://github.com/fgagamedev/Traveling-Will>

Description: This game tells the story of Will, personification of the Will, trying to restore

Remarks: This game has one of the most attractive user interfaces from the games packaged so far. The team that developed it was able to create a very good game, technically speaking, with engaging scenarios and soundtrack, because they had design and music students. A screen of the game running after compiling it with the building script is shown in Figure 10.

Figure 10 – Traveling Will

B.17 Deadly Wish

Authors: Lucas Mattioli, Victor Arnaud, Vitor Nere, Iago Rodrigues

Year: 1/2016

Genre: Battle Arena

 # Players: Single player

Repository: <https://github.com/fgagamedev/Deadly-Wish>