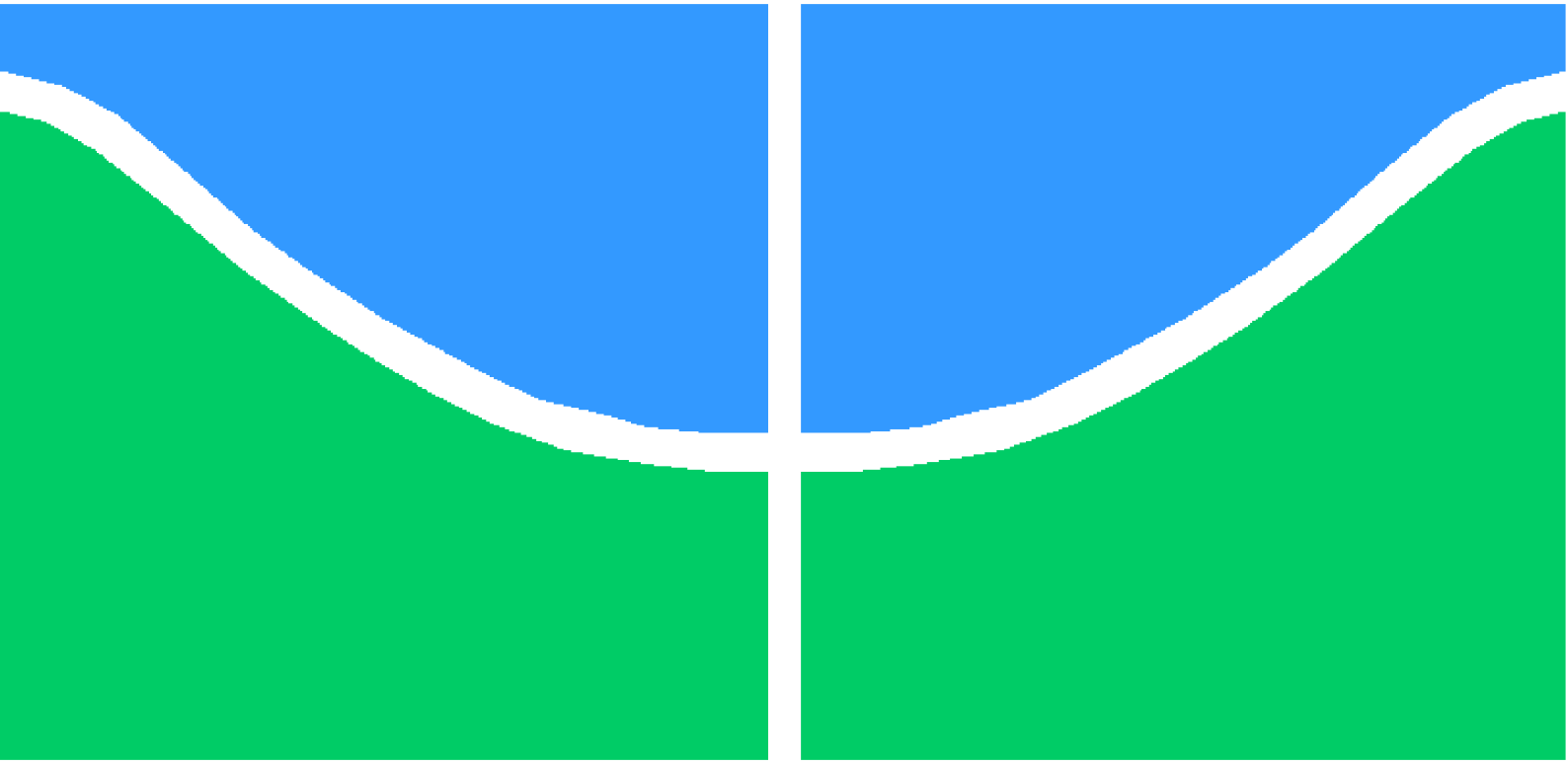

Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia de Software

UnB Games Platform: A collaborative project

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

Brasília, DF
2017

Parley Pacheco Martins

UnB Games Platform: 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

Brasília, DF

2017

Parley Pacheco Martins
UnB Games Platform: A collaborative project/ Parley Pacheco Martins. – Brasília, DF, 2017-
55 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 Platform: A collaborative project

CDU 02:141:005.6

Parley Pacheco Martins

UnB Games Platform: 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, 05 de julho de 2017:

Prof. Dr. Edson Alves Da Costa Júnior
Supervisor

Prof. Dra. Carla Silva Rocha Aguiar
Guest 1

Prof. Matheus de Sousa Faria
Guest 2

Brasília, DF
2017

Acknowledgements

Caso não deseje utilizar os agradecimentos, deixar toda este arquivo em branco.

A epígrafe é opcional. Caso não deseje uma, deixe todo este arquivo em branco.

Resumo

Jogos desenvolvidos nas universidades não possuem muito reconhecimento ou suporte. Usuários finais nunca têm a chance de jogar ou dizer qualquer coisa, como críticas ou elogios, 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 púbilco, porque o trabalho realizado numa universidade, especialmente púbilca, deve ser acessível a todos e porque esses jogos foram feitos com o propósito de promover diversão e aprendizado para os usuários. Este documento descreve como este trabalho será feito, criando a plataforma para o compartilhamento destes jogos.

Palavras-chaves: jogos. desenvolvimento. plataforma.

Abstract

Games developed in the University don't have much recognition or support. End users usually never get to play or say anything about any version of any of them, either good or bad ones. Most people don't even know that games are created in classes. This project aims to make these games available to people, because the work made in a university, especially a public one, should be accessible to everyone and because these games were made with the purpose of learning and providing fun to the users. This document outlines how this work will be achieved, by creating a platform to upload the created games.

Key-words: games. development. platform.

List of Figures

List of Tables

List of abbreviations and acronyms

SDL 1	Simple DirectMedia Layer version 1
SDL 2	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*

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* 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 goals of this work are the following:

- create an on-line platform to host the games developed in the courses of this University;
- 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.

1 Basic Concepts

This chapter gives an overview on 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 Linux Filesystem, that helps developers to understand where their binaries and other files should go on the user's system. At 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). The basic assumption of a game is that it provides a fun time, bonding and usually learning of new skills. Even though there are many games meant to play alone, they normally are meant to play with someone else (even if that someone is the computer), because the interaction with something is a key concept for any game.

They can take several format, like board, card, dice games. Each format has unique strategies to win. Board games usually divide the user space in sectors, and everything is related to which sectors you are in and how you control them. Card games are related to symbols (the cards) and the possible combinations of them (CRAWFORD, 1984).

Since computers were invented they completely changed the gaming world. New kind of games, like *first person shooter* and *tower defense*, were created and made popular. With the Internet, it became even easier to own and play different games, also it's possible to play almost any game with anyone in the world.

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

1.2 SDL

Simple DirectMedia Layer (SDL) is a library that helps developers by creating cross-platform APIs in order 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). In order 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 (LAZYFOO, 2017).

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 running time. On Linux systems they have the *.so* file extension, while on Windows they have *.dll* (HOWTO, 2017). 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 in the user's system, though.

The static library is compiled against the main source code and it's merged to 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 bigger binary. The new license on SDL 2, *zlib*, allows users to use SDL as a static library, however they are not encouraged to.

1.3 Filesystem Hierarchy Standard

The Filesystem Hierarchy Standard 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 (ALLBERY et al., 2015).

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 changed by regular users (ALLBERY et al., 2015)

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 on 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, p. 3). The `/usr` hierarchy should include shareable data, that means that every information host-specific should be placed in other directories.

About the `/var` hierarchy, FHS specifies that "everything that once went into

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

`/usr` that is written to during system operation (as opposed to installation and software maintenance) must be in `/var`." (ALLBERY et al., 2015, p. 30).

1.4 Repository

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, 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" (CHACON; STRAUB, 2014). 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 (CHACON; STRAUB, 2014). This feature lets developers create and test new changes before submitting them to the project stable line of work, without affecting the final product.

1.5 Packages

In computer science package can have multiple meanings, depending on the context being used. A Linux package means a bundle of files containing the required data to run an application, such as binaries and information about the package.

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.

1.5.1 CMake

Creating packages for multiple platforms requires a lot of time and effort, because it has to be compiled on each of the systems, with those system's libraries, binaries and architecture. In order to make this task easier, several cross-platform building tools were created.

CMake was created to fulfill the need for "a powerful, cross-platform build environment for the Insight Segmentation and Registration Toolkit" (CMAKE, 2017). It is "a system that manages the build process in an operating system and in a compiler-independent manner" (CMAKE, 2017).

In a very clever way, CMake generates native compiling and configuration scripts (like makefiles for Unix and namespaces for Windows) and use them to build the package (CMAKE, 2017). It is also designed to be used with the native environment, unlike other building tools (CMAKE, 2017).

The building process is controlled by files named `CMakeLists.txt`. They can have commands to generate a building environment, set needed variables, compile dependencies, link required libraries and install the project. A project that has many subdirectories, each with their own rules, can have multiple `CMakeLists.txt` to create the final product.

2 Methodology

This chapter explains what was and will be done within the duration of the whole project. Section 2.1 describes the overview of the whole project. Section 2.2 elucidates how the tasks were divided among all the involved for this first part of the project, while 2.3 shows how and which games were selected to first test the scripts. Section 2.4 demonstrates the folder structure to create the packages. Section 3.2 talks about the development of the platform. The chosen tools for the project are specified in 2.6

2.1 Project Overview

It has the main goal of creating a platform with 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 a single package that runs in Linux distributions without the need of installing any other package; they also must have a graphical installer for users without technical knowledge.

In order to achieve this goal, the games developed will be cataloged and cloned to a main GitHub organization (whenever possible). Two scripts will be created then, one to build games using SDL 1 and the other for SDL 2. The platform itself will be developed while all the other activities take place.

After that, a script will be generated to replicate the packaging system to all of the other games, making the necessary adjustments along the way. The games will be deployed to the website with all of their information and available installers.

The packaging scripts will be integrated and adapted to the platform, so that any student who posts a game will have the installers generated automatically.

2.2 Task Division

This project is totally collaborative, it depends and relies on different classes and courses. Because of that, 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 will remain as helpers in the packaging system and main stakeholders for the team developing 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* (Man-

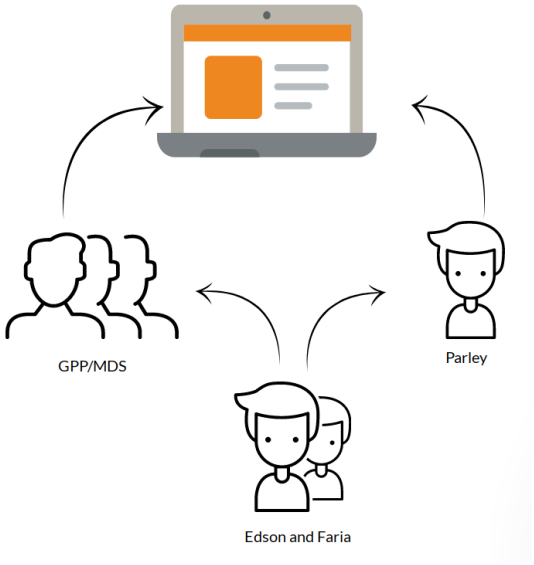

Figure 1 – Task Division

agement of Portfolios and Projects) is in charge of creating the first version of the actual website with some of the features desired.

The scripts creation and their application on the games, their integration with the platform, as well as the evolution and maintenance of the platform after the courses are finished will be my responsibility.

2.3 Game Gathering

The games selected for this 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. Because this work is 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, that was ministering the course until last year, and Mr. Matheus Faria, the new teacher, first contacted the students and asked them to post their codes to GitHub. They cloned them into the `fgagamedev`[1] 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.

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

Table 2 – Initial status of the selected games

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

Out of 20 games created in *Introdução aos Jogos Eletrônicos*, 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.

2.4 Packaging

To create the installers for the games, professor Edson decided to use a folder structure that would be easy to understand to anyone familiar with Linux. Apart from the original directories in the repository, he added the folders `bin`, `dist`, `lib` and `linux`.

- `linux` would contain the scripts needed during compilation and also helpers to run the game after its building. When other OSs are added, there will be folders with the specifics for each one of them;

- `lib` initially consists of the source of the libraries used by the games. They are built inside this folder as well, according to the Operating System the package is being built for;

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

- `dist` includes the final installers, according to each supported OS. These can be distributed and installed in any supported computer;

- `bin` has the compiled libraries and the game executable.

I added the directory `Qt` to use the Qt Installer Framework without having to install it system-wide. Figure 2 shows the folders and their contents to a SDL 1 project.

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 was 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. The most experienced members also helped the others to learn the technologies to develop the website.

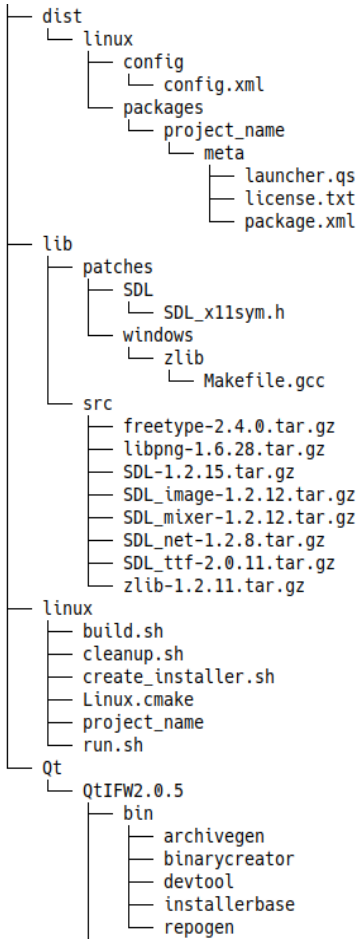

Figure 2 – Folder tree

2.6 Tools

CMake is the chosen framework for generating the packages. It's suppose to help developers creating applications that run in several platforms, like Linux, Mac and Windows. It offers a lot options for that, like cross compilation and compilation directed to each of them. It is distributed under OSI-approved BSD 3-clause License and the minimum required version is 2.8.

For the graphical installer, Qt Installer Framework has been selected. It is easy to use and offers a nice GUI with all the necessary steps for installing a package, like license agreement and path choice. This framework is distributed under LGPLv3 license and the version being used is 2.0.5.

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 on 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.

Python is the language for the packaging script because it must integrate with the Django webapp and its powerful easy to use API. It will also be the language for any other needed scripts. The least required version is 3.4, and it's licensed under an OSI-approved open source license.

To develop the scripts, a virtual machine running Debian Jessie was used. The VM was powered by Vagrant, version 1.9, 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 script and used to its development has an Intel Core i5-6200U 2.3 GHz processor. It also has 8 GB of RAM and an NVIDIA GeForce 940M graphic processor.

3 Partial Results

This chapter explains the results obtained so far with the project development. Section 3.1 shows the results of the building scripts and the problems to create them, while Section 3.2 explains the same about the platform. Section 3.3 outlines some problems, from both the scripts and the platform, that have to be fixed in the next part of the project.

3.1 The Building Scripts

The building scripts were tested against half of the 16 available games. Table 4 shows which games were tested and the results of them. It was a success, because all the installers were created correctly for all of them (both GUI/Qt and *.deb*). For some games, even with the correct compilation, they wouldn't run as expected, due to logical errors in their source code.

The game *Space Monkeys* compiled correctly, however upon running it, the user couldn't do anything and the screens weren't precisely rendered. *War of the Nets*, even compiling without errors, had a segmentation fault after a few seconds with the game open.

Table 4 – Scripts results

Game	**Compiles?**	**.deb**	**GUI/Qt**	**Runs?**	**SDL**
Ankhnowledge	y	y	y	y	1
Post War	y	y	y	y	1
Jack the Janitor	y	y	y	y	1
Emperor vs Aliens	y	y	y	y	1
Ninja Siege	y	y	y	y	1
Space monkeys	y	y	y	n*	1
War of the Nets	y	y	y	n	2
Travelling Will	y	y	y	y	2

The building scripts are very similar, the only difference is that one builds games for SDL 2 while the other uses SDL 1 (and their respective libraries). They work by cloning the repository and copying the required files to use CMake as seen in Algorithm 1. This is the tool that actually compiles and builds everything. Because CMake generates a lot of files that are only used while it's running, professor Edson made a few more scripts to separate everything into folders and generate the installers.

Algorithm 1 Algorithm to build the games

```
Start
    project ← INIT(url, branch)          ▷ Clone repository and set the project
    COPY_FILES(default, project)                          ▷ Copy templates
    project.media_dir ← FIND_MEDIA                  ▷ Find the media folder
    project.source_dir ← FIND_SOURCE   ▷ Find the source folder and .cpp files
    REPLACE_INFO(default, project)                  ▷ Replace template defaults
    RENAME(default, project)                              ▷ Rename some files
    BUILD(project)                                     ▷ Call the build script
    CREATE_INSTALLERS(project)                        ▷ Create the installers
End
```

A major concern when making this script was its generality. It should run successfully with as many games as possible, requiring only a few tweaks in the source code or folder structure of the repository, if any. Starting with *Jack the Janitor*, the example Professor Edson had made first, the building script assumed a lot of things, mostly due to my inexperience with games and the folder structure adopted by the students. For example, in the initial versions, I thought all the *.cpp* files would always be in a folder called `src`. Another assumption was that all media would be in a `media` folder.

Both of them proved me wrong as more games were tested, but were fairly easy to fix and keep the algorithm generic. For both folders, I had to modify the script to look for directories that would have similar names to those I thought were the rule. For example, `source` instead of just `src`, and `resources`, `res` and `sound` instead of just `media`. Even though this doesn't find all possible names, it follows a pattern found in most folder structures and all of the projects tested so far.

Another big premise was that all games would have their main file named `main.cpp`. Even in repositories with Portuguese file names, it never occurred to me that anyone would name those files in any other way. However, a few games, specially *Traveling Will* proved me wrong. Here, again, there was the option of looking for files named with similar words to `main`, but this was a terrible option in this case, because the file could have *any* name. Even if I looked for `principal.cpp` that would not guarantee anything.

One other option was parsing all the source files looking for the main function, but that would slow down the building process and would be error prone. Because there were so many options, I decided to keep the script looking for `main.cpp`, even if it required to manually change the repository.

3.2 Platform

The team developing this first version of the platform was able to integrate Django and React with some difficulty. Both of the technologies chosen are very well established

on their own. Putting them together, however, is another matter, where they had a real hard time to make everything work right. They used React as the main user interface and Django Admin package to create the administrator part of the game.

The website as of now allows an administrator to upload a game, with its respective information, like supported platform, related media, and installers. The administrator has to manually add all the information related to a game, like developers who worked on it, awards won (if any), release date, version number, etc. Figure 3 shows part of the screen to add a game.

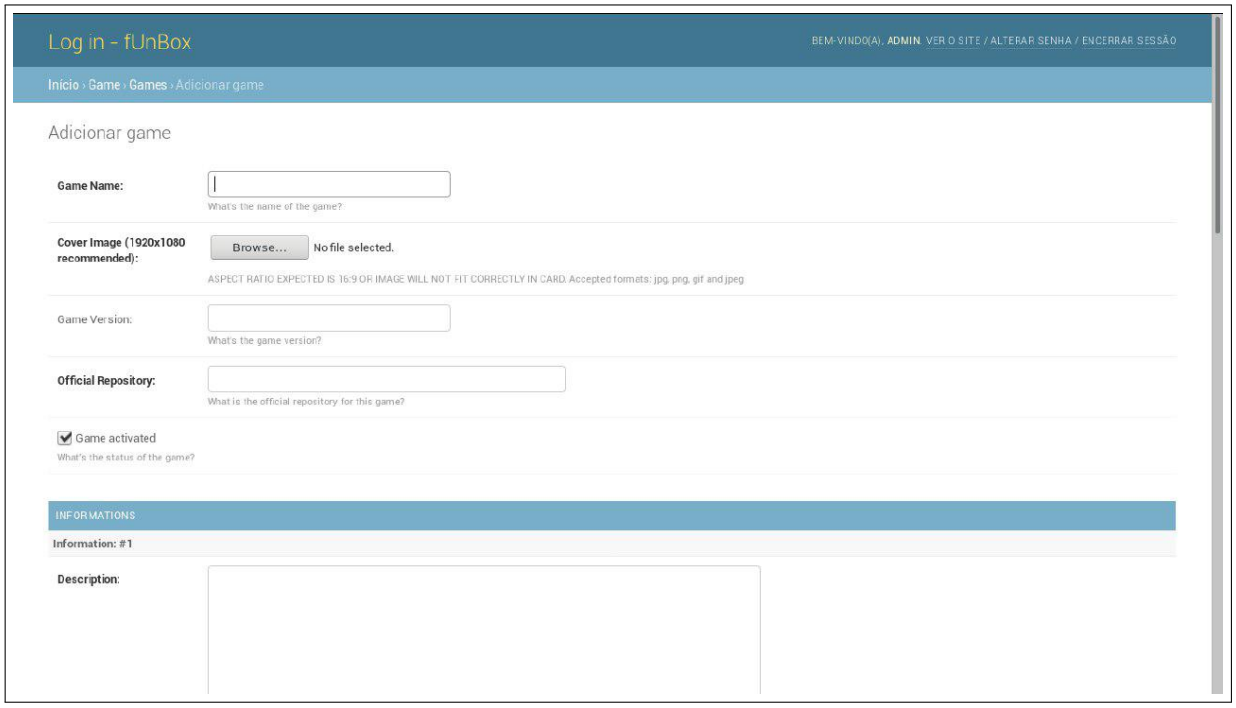

Figure 3 – Include new game

The general public can see a list of the available games, with their uploaded pictures. The home page also shows a slide with some pictures of highlighted games. By choosing one, it's possible to see its version, official repository, release date, description, among other information.

Some other features of the platform are the possibility to download the game for the available operating systems or comment using a Facebook account, as shown in Figure 4. It's also possible to categorize the games and apply several filters on them. The user can search for a specific game by its name or description as well.

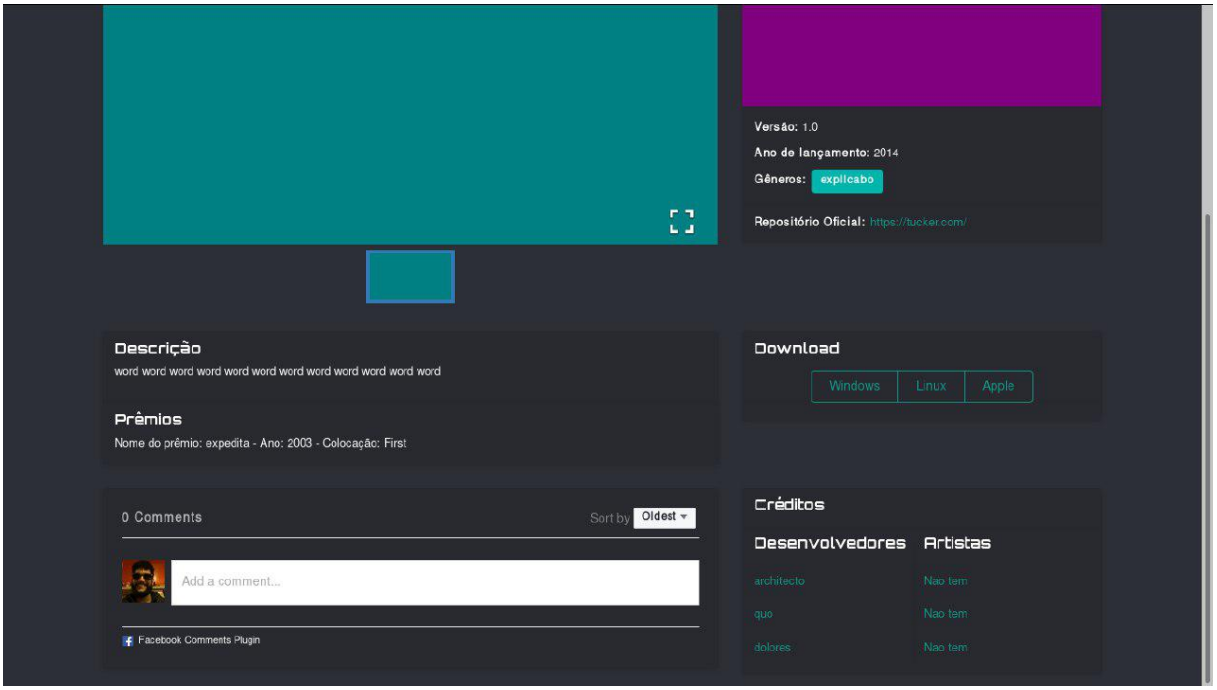

Figure 4 – Game detail

3.3 Known Issues

The building scripts have a few problems that will be fixed in the second part of the project:

- choosing a destination folder different than the user's home directory causes the Qt installer to fail. This is likely happening because somewhere on the Qt templates the symbol ~ (abbreviation for home) is being used;
- when the installation works successfully, the game doesn't run properly (or at all) on some systems (tested on Arch Linux for now). Probably there are some missing libraries on the final package, but this is just a hunch and has to be proved;
- the games are running without sound, even on host systems that have full access to the sound card.

The platform also has a few issues that weren't resolved due to lack of time of the team. Some of them are described below:

- when an administrator tries to save a game without one or more needed fields an error occur, but all data is lost, requiring them to reenter all the information they had already typed. This may be happening because in the form validation, the team

might be sending a new instance of the object instead of the one with the validation errors;

- images are being saved multiple times when filling different forms (of the same game). They think this may happen because they retrieve and save all forms instead of just one with the images;
- kernel choice is hard coded on the system;
- when reporting a bug, the request breaks on some web browsers, like Chrome. They think it must be something with the protocol being used (HTTPS, for Chrome, and HTTP for Firefox).

4 Future Work

This chapter explains what will be done in the remaining time of the project. It talks about some of the long term goals and the activities that will be carried to achieve them.

4.1 Schedule

For the next months it is expected to have the script for SDL 2 projects finished with support for Windows and MacOS. The known issues described on Section 3.3 have to be fixed for both SDL 1 and 2. The building scripts have to be tested against all the available games.

Another goal is to allow the user to link their GitHub repository to the website, letting them build the packages for the game automatically with GitHub hooks. The game will then, be available in the website without the need for a manual upload from an administrator.

It's also purpose of this next part of the semester to maintain and evolve the platform while the tasks related to the script are being held. Some selected issues will be resolved to add new features and fix bugs found on the system.

The following activities will be developed in the remaining time of the project. They are summarized in Figure 5.

1. **Literature Review**: review what the literature has on packaging, CMake, game development.

2. **Add Lua support**: some of the games, especially the ones built with SDL 2, have lua as a dependency library. This lib is not being being compiled in the current version of the building scripts.

3. **Add other Linux distros support**: allow other users to install the games using their package manager. Generate at least *.rpm* packages in addition to the *.deb* already generated.

4. **Add Darcy's games**: look for games developed at Darcy Ribeiro campus and run the building scripts on them.

5. **Add MacOS support**: create install packages for Mac (Apple Systems).

6. **Add Windows support**: make an installer (*.exe*) to run on Windows 10 (maybe with some backwards compatibility if possible).
7. **Integrate to platform**: run the scripts though a request on the website
8. **Avoid duplication on save**: according to the GitHub repository of the team, models are being saved with duplicated data.
9. **Create games statistics**: create endpoints to send game statistics, like views and download amounts.
10. **Submit issues to the GitHub repository**: allow users to submit issues through the platform to the GitHub repository of the game.
11. **Integrate to GitHub**: make the system receive GitHub signals.
12. **Allow user to build a game from a branch**: et the user upload a game based on a chosen branch. This will in fact run the script to build the game, but without the need of a an administrator.
13. **Read game info from game branch**: developers, description and even some images are usually available in the game repository. This task is to read that information to avoid the need of manual input of the administrator.
14. **Code refactoring**: integrate scripts (SDL 1 and 2), make them more generic and efficient.
15. **Final adjustments**: make minor improvements and fixes.
16. **Write Report**: report progress and results.

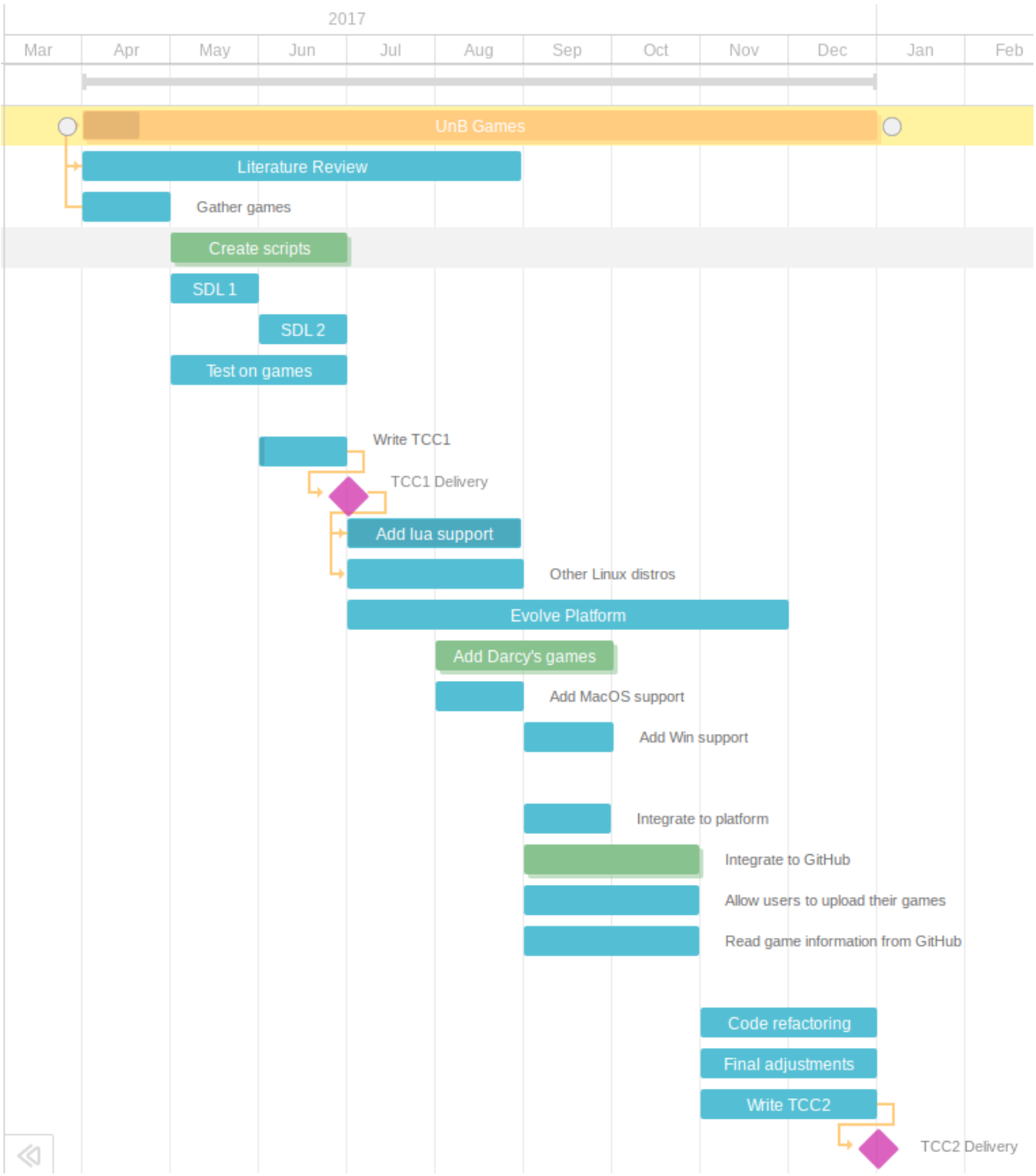

Figure 5 – Project Schedule

Bibliography

ALLBERY, B. S. et al. Filesystem hierarchy standard. 2015. Cited 3 times on pages 15, 24, and 25.

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 23.

CHACON, S.; STRAUB, B. *Pro git*. [S.l.]: Apress, 2014. Cited on page 25.

CMAKE. *CMake Overview*. 2017. Disponível em: <https://cmake.org/overview/>. Cited on page 26.

CRAWFORD, C. *The Art of Computer Game Design*. Berkeley, CA, USA: Osborne/McGraw-Hill, 1984. ISBN 0881341177. Cited on page 23.

HOWTO, P. L. *Shared Libraries*. 2017. Disponível em: <http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html>. Cited on page 24.

LAZYFOO. *SDL Tutorials*. 2017. Disponível em: <http://lazyfoo.net/tutorials/SDL/>. Cited on page 24.

LINODE. *Linux Package Management*. 2017. Disponível em: <https://www.linode.com/docs/tools-reference/linux-package-management>. Cited on page 26.

PRESSMAN, R. *Software Engineering: A Practitioner's Approach*. 7. ed. New York, NY, USA: McGraw-Hill, Inc., 2010. ISBN 0073375977, 9780073375977. Cited on page 23.

SDL. *Introduction to SDL 2.0*. 2017. Disponível em: <https://wiki.libsdl.org/Introduction>. Cited on page 24.

WEBSTER, M. *Definition of repository*. 2017. Disponível em: <https://www.merriam-webster.com/dictionary/repository>. Cited on page 25.

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

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

B.3 Ninja Siege

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

[1] Available on the game repository: https://github.com/fgagamedev/Jack-the-Janitor

Year: 2/2012

Genre: Tower defense

 # Players: Single player

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

B.4 Space Monkey

Authors: Victor Cotrim

Year: 2/2012

Genre: Tower defense

 # Players: Single player

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

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 6 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.

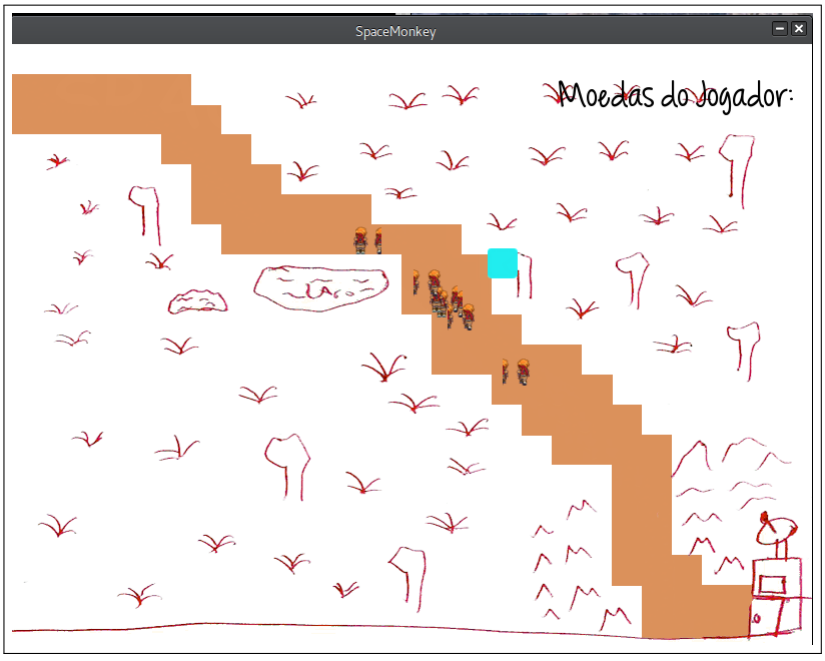

Figure 6 – Space Monkey

B.5 War of the Nets

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

Year: 2/2013

Genre: Turn Based Strategy

 # Players: Multi player

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: Multi player

B.7 Ankhnowledge

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 7.

Authors: Arthur del Esposte, Alex Campelo, Atilla Gallio

Year: 2/2013

Genre: Turn Based Strategy

 # Players: Multi player

B.8 Last World War

Authors: Gabriela Navarro

Year: 2/2013

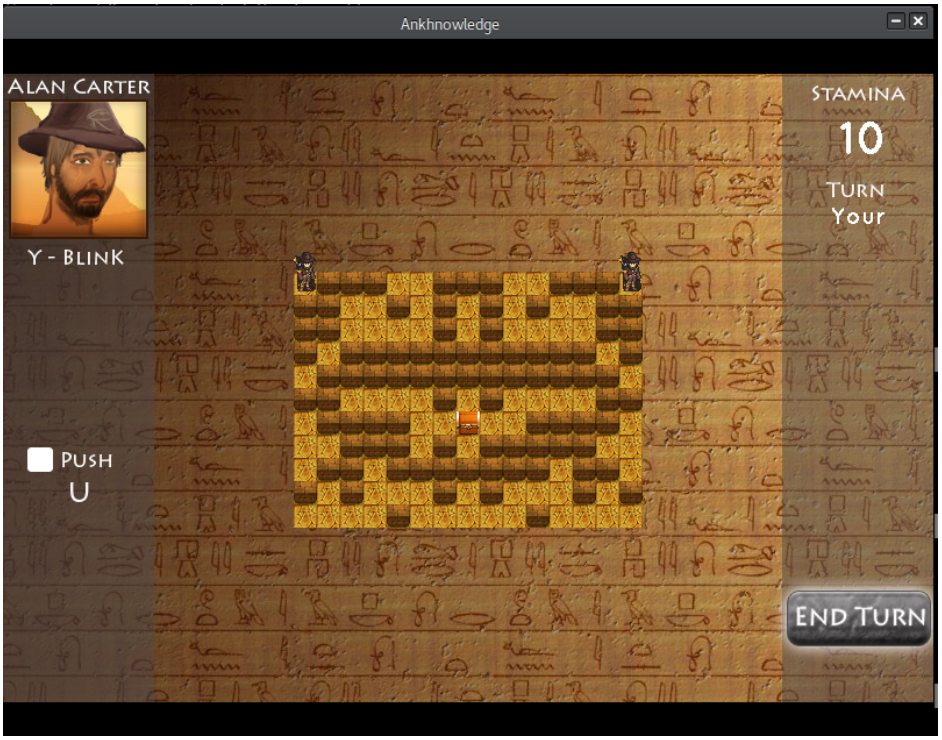

Figure 7 – Ankhnowledge

Genre: Turn Based Strategy

 # Players: Multi player

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:

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:

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.

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

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

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

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

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

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

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 8.

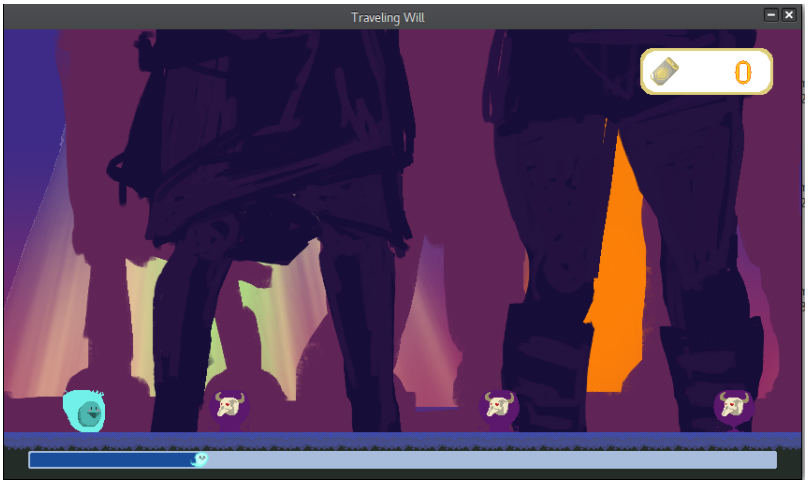

Figure 8 – Traveling Will

B.17 Deadly Wish

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

Year: 1/2016

Genre: Battle Arena

 # Players: Single player