**Bilkent University**

Department of Computer Engineering

**CS 319 Project**

*JCrawl: 2D Top-down Adventure Game*

Final Report

Group Members

* Arda Yücel
* Cheol Woo Park
* Fatih Taş
* Mustafa Fidan

Contents

[1. Implementation 3](#_Toc449718189)

[2. Status Report 3](#_Toc449718190)

[3. User Guide 4](#_Toc449718191)

[3.1 Introduction 4](#_Toc449718192)

[3.2 Installation 4](#_Toc449718193)

[3.3 Playing the game 4](#_Toc449718194)

[3.4 JCrawl Level Design Language - JLDL 4](#_Toc449718195)

# 1. Implementation

It was a given that there would be challenges in trying to implement this kind of game. Especially since our group decided to merge two different collision grids into one, introducing an entire new layer of complexity which had to do with converting arbitrary pixel values into array grid and vice versa.

Therefore, the biggest challenges concerning the implementation was the collision handling. Many more private/protected methods and variables had to be introduced as our design was not adequate at tackling these issues by its own. Also, we did not anticipate that our Entity class would be the heaviest amongst all classes present in our package in amount of code wise.

Another obstacle to implementation was surprisingly the file parsing part of the implementation. As we chose to encapsulate our data into data storage classes, it made it extremely difficult to look into the data for debugging purposes, but it made it easier when it came to implementing features that utilized those data.

We solved these issues by employing every bit of experience, knowledge we had. One trend we realized was that the graphic/rendering part of the project was actually the easiest to implement and had least amount of bugs.

Fortunately, no major changes were made to our design. (Only 2~3 new classes were added and those were completely for utility. E.g. ImageLoader and SpriteSheet classes) The only major design change that was involved was actually just splitting existing classes into more smaller level. Also, the most successful method in tackling technical challenges were indeed the “divide and conquer”. One example would be where we had severe calculation errors occurring in our pixel collision, and we broke down the problem into smaller parts, isolating the problems into private methods, which helped us to solve the problem (partially).

# 2. Status Report

* Render System – 100%
* Game Mechanics Management – 100%
* Input Management – 100%
* External Data Management – 100%
* Game Event Management – 100% (90% if we take into account of bugs)
* Gameplay Object Management – 40%
* Level Management – 100%
* Miscellaneous (Data Storage Classes) – 100%

Looking at the following, you can see that Gameplay Object Management is the most lacking component in our project. However, given that we did finish most of the framework, with little bit more time, completing rest of incomplete features shouldn’t be a hassle at all.

# 3. User Guide

3.1 Introduction

JCrawl is a top down 2d adventure game, with the emphasis placed on extendibility and user level design. The aim of the game is to complete all levels by defeating every other monsters existing in the level.

3.2 Installation

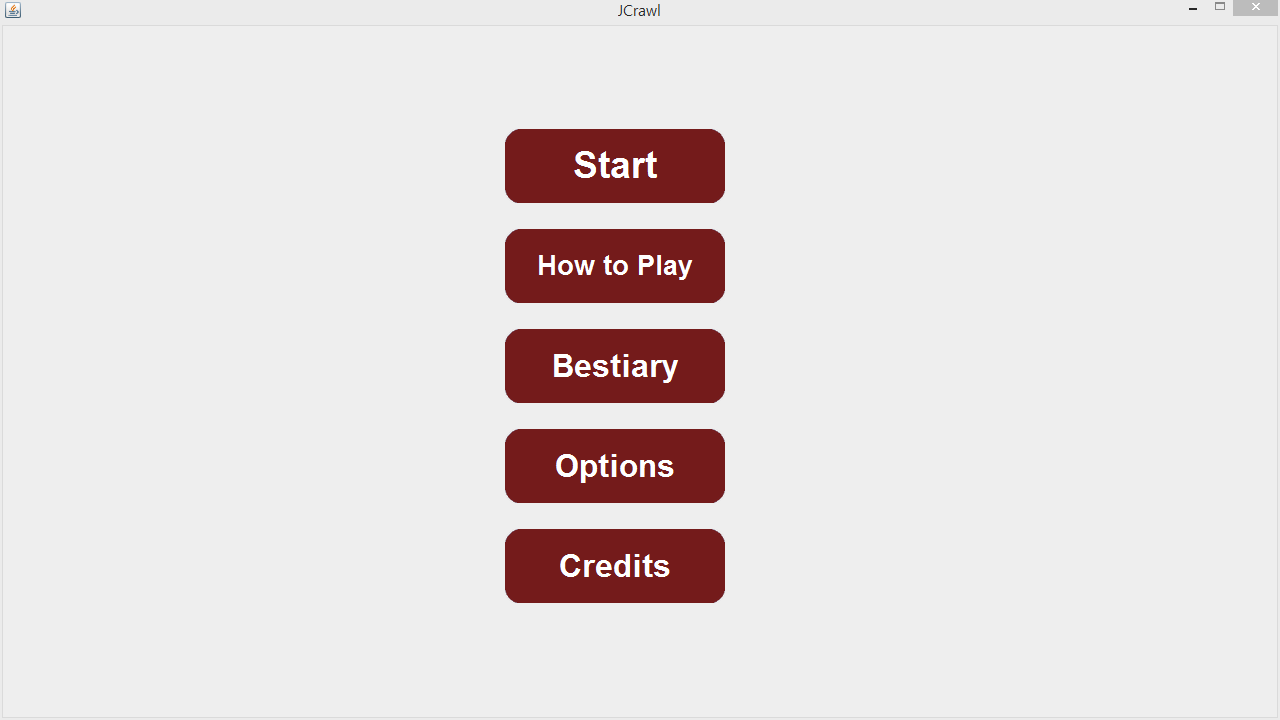
1. System Requirements

* Your PC must be able to run the latest Java Runtime Environment
* It can run on both 32 bit and 64 bit operating systems.
* Recommended Resolution is 1024x768. But the game setting also supports 800x640.

1. Installation

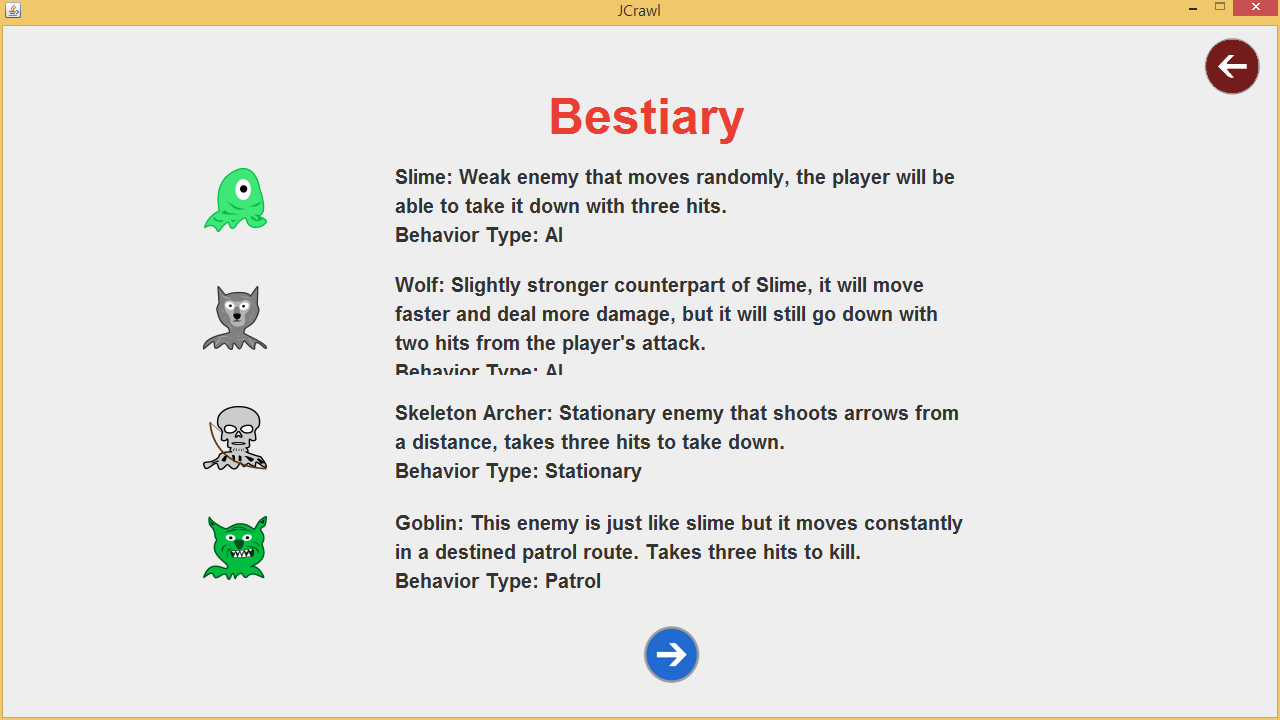
Unzip the Game.zip and run the JCrawl.jar to play the game.

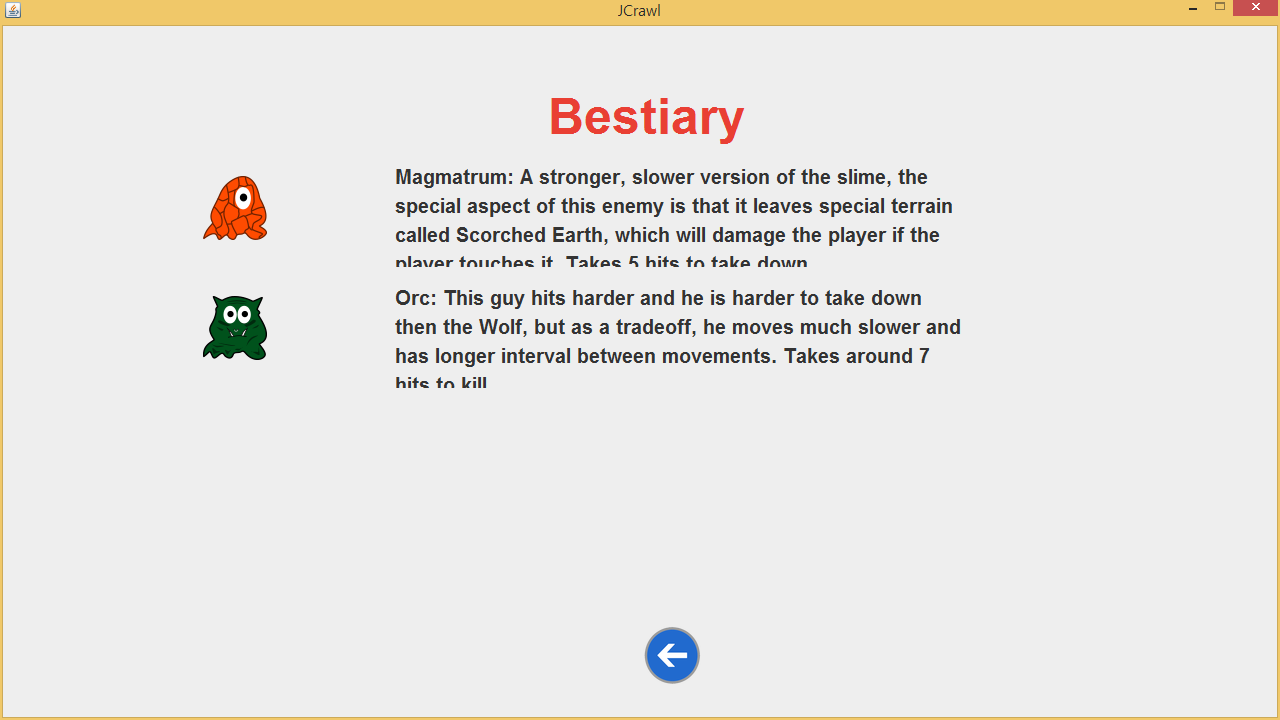
3.3 Playing the game

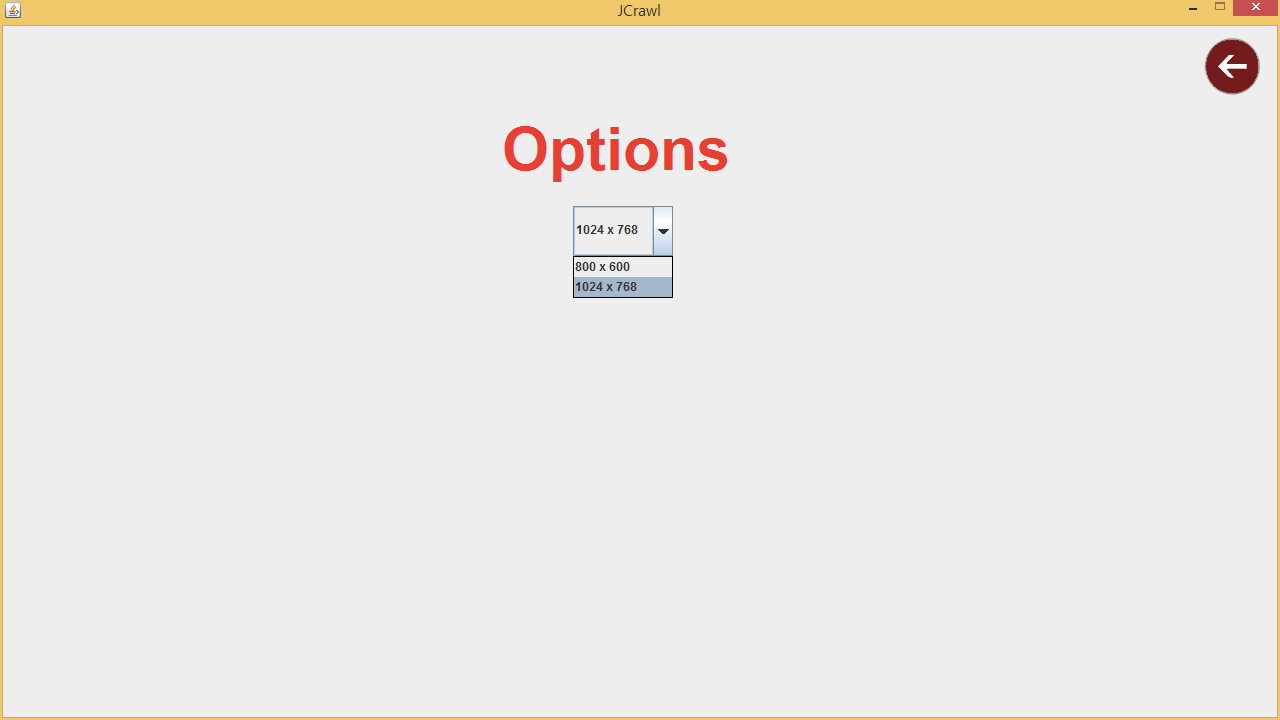
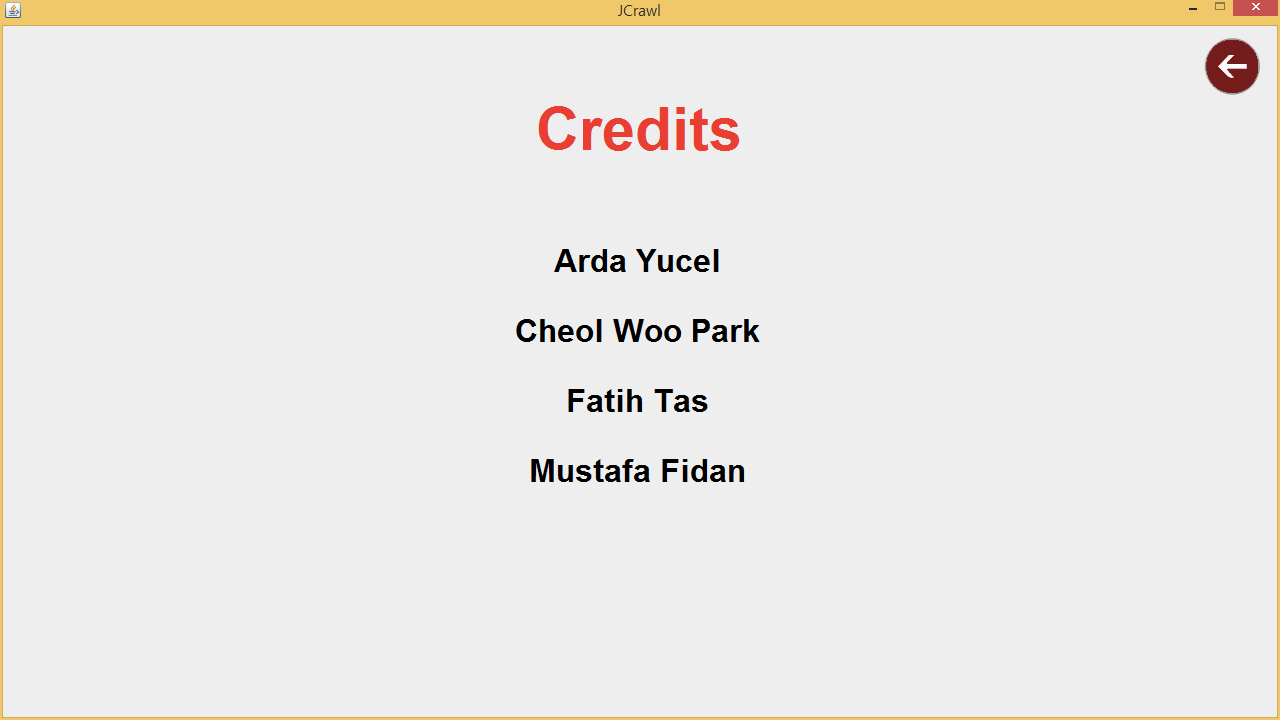
* Main Menu

1. Start



1. How to Play
2. Bestiary



1. Options
2. Credits

* Gameplay

Refer to “How to Play” and “Bestiary” in-game to obtain more information.

3.4 JCrawl Level Design Language - JLDL

JCrawl provides the user the ability to create, edit existing levels to extend their gameplay experience.

Inside the res folder which was unzipped with the JCrawl.jar, you will find multiple test files. By editing the text files you will be able to edit the levels.

Here is the order in which the levels are parsed:

level.txt -> level1.txt -> level2.txt -> ….

The first level is always just plain level.txt file.

1. **General layout**

[layout]

0,0,0,0,0,0,1,1,1

…

[entities]

p\_, x, y

type, name, x, y, startx, starty, endx, endy

type, name, x, y

You must have [layout] as the first token of the level.txt file, or else it will fail to run.

The following texts are the code for the level design itself. JCrawl’s tile grid’s dimensions are 16x12, so, you will need 12 lines of 16 tiles, each separated by comma. **There must not be any space between commas**.

1. **Level definition**

Sample: 0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0

This will create a row of tiles where there will be a wall on the ends and rest will be empty.

0 – Wall

1 – Walkable

1. **Entity Defintion**

The very first definition must be a player. This is the format:

**p\_,x,y**

Where x is the position in the x axis and the y is the position in the y axis.

And here are the formats for the rest.

**AI,type,x,y**

This code will generate randomly roaming creature types at the coordinate (**x,y**) which is of **type**

Here are the list of valid AI types:

1. **slime**
2. **wolf**
3. **orc**
4. **magmatrum**

**stationary,type,x,y,dir**

This code will generate stationary creature looking at defined direction **dir** at coordinate (**x,y**)which is of type **type.**

**Types –** only **archer** for this version

**dir –** 0 : up, 1 : left, 2 : down, 3 : right

**patrol,type,x,y,x\_t,y\_t**

This code will generate patrolling creature, using the path made by **x,y** and **x\_t,y\_t** with type **type.**

**Disclaimer**: The pathing only supports straight path, so either **x** and **x\_t** or **y** and **y\_t MUST BE EQUAL**.

**Types –** only **goblin** for this version.