**Bilkent University**

Department of Computer Engineering

**CS 319 Project**

*JCrawl: 2D Top-down Adventure Game*

Analysis Report

Group Members

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

Contents

[1 Introduction 3](#_Toc446765070)

[2 Overview 3](#_Toc446765071)

[3 Gameplay Elements 3](#_Toc446765072)

[4 Functional Requirements 6](#_Toc446765073)

[5 Non-functional Requirements 6](#_Toc446765074)

[6 Pseudo Requirements 7](#_Toc446765075)

[7 System Models 7](#_Toc446765076)

[7.1 Use Case Model 7](#_Toc446765077)

[7.2 Dynamic Models 13](#_Toc446765078)

[7.3 Object and Class Models 19](#_Toc446765079)

[7.4 User Interface – Navigational Paths and Mock up Screens 20](#_Toc446765080)

[8 Glossary 28](#_Toc446765081)

[9 References 28](#_Toc446765082)

Analysis Report

JCrawl: 2D Top-down Adventure Game

# 1 Introduction

JCrawl is a purely Java based top down adventure game, the basic game mechanics/fundamentals of the game such as room exploration, movement, and combat will be inspired from the critically acclaimed game called Binding of Isaac, but with some design ideas borrowed from The Legend of Zelda, such as the UI design, art style and misc. non-functional related aspects.

**Binding of Isaac link:**

<http://store.steampowered.com/app/113200/>

**Legend of Zelda link:**

<https://en.wikipedia.org/wiki/The_Legend_of_Zelda_(video_game)> (as the game was released in 1996, the link will be connected to a Wikipedia page about it)

The primary focus of this game will be heavily on Player-to-Object and more specially, Object-to-Object interactions like pressing a switch to turn on the lighting, killing enemies by shooting projectiles at them, moving blocks on a top of a pressure plate to keep the traps off and such.

# 2 Overview

The player controls a single character to explore myriad of rooms filled with dangers, loot and puzzles. The player will be able to move freely in all directions, but shoot at the direction they are looking at only initially. There will be many kind of items which will further enhance player's capabilities (Temporary power ups/ Permanent upgrades). Some enemies will be stationary, some enemies will have hard coded behavior, and some enemies will have independent basic AI. There will be two types of objects: Independent/Dependent; Independent objects (e.g. Fire traps, Automatic doors) will perform their intended behavior regardless of whether the player input is present or not. Dependent objects (e.g. Light switch, Doors) will perform their intended behavior IF it receives a player input.

The level design will be editable by user, allowing for further replayability, the documentation containing required instruction on how to do so will also be shipped with the final product, and the only tool required to perform level editing will be any kind of text editor like Notepad or Word.

# 3 Gameplay Elements

* **Player**

The player controlled character will possess variety of stats that the user can upgrade and utilize throughout the course of the game: Health, Movement Speed, Attack Speed, and Damage; Picking up certain upgrades will either permanently boost its stats. For example, picking up an item named “Rapid-fire Module” will permanently alter player’s interval between each attack so that it is more rapid.

The player can also obtain completely new game mechanics with certain items, e.g. ability to push around certain blocks, immunity to flame damage.

* **Enemies**

Enemies will have three different types of behavior set to them: Rudimentary AI, Hard coded behavior, or they are stationary. Rudimentary AI will cause enemies to roam around aimlessly, hard coded behavior will make them move in predetermined manner (so they look like patrolling certain areas for example), or they will be stationary and attack the player from range.

1. Slime: Weak enemy that moves randomly, the player will be able to take it down with a single hit.

Behavior Type: AI

2. Wolf: Slightly stronger counterpart of Slime, it will move faster and deal more damage when coming in contact with the player, but it will still go down with single hit from the player’s attack.

Behavior Type: AI

3. Orc: This guy hits harder and he is harder to take down then the Wolf, but as a tradeoff, he moves much slower and has longer interval between movements. Takes around 5 hits to kill.

Behavior Type: AI

4. Goblin: This enemy is just like slime but it moves constantly in a destined patrol route. Only takes one hit to kill.

Behavior Type: Hard coded

5. Skeleton Archer: Stationary enemy that shoots arrows from a distance, only takes one hit to take down.

Behavior Type: Stationary

6. Magmatrum: A stronger, slower version of the slime, the special aspect of this enemy is that it leaves special terrain called “Scorched Earth”, which will damage the player if the player touches it. Takes 5 hits to take down.

Behavior Type: AI

7. Final Boss (Placeholder name): This enemy will move fast, and unlike all other enemies (except stationary types), will actively track the player and attack him from distance with projectiles just like the player. Takes 20+ hits to take down.

Behavior Type: AI

**Disclaimer:** Any specific numbers associated with the enemy’s stat is **not** final and will always be subject to change in the Testing phase.

* **Objects (Independent)**

Independent objects execute their behavior with/without player’s intervention. For example, dart trap will constantly shower certain area with darts regardless of whether the player is in the area or not. However, this does not mean player cannot interact with the independent objects in a way that changes their behavior (e.g. pressing a button shuts down all the traps in the room)

1. Dart Traps: Shoots darts at the fixed direction.

2. Spike Traps: Melee version of Dart Trap

3. Fire Traps: A hybrid between Dart Trap and Spike Trap, while the range isn’t melee, but it doesn’t exactly reach from the other end of the room to the other end as well. The range of this trap will be around 3~5 Tiles.

4. Automatic Doors: A door that closes and opens with set interval.

5. Monster Spawner: Spawns AI controlled enemies with long intervals between the spawn.

* **Objects (Dependent)**

Dependent objects, contrary to independent objects, will not execute their predetermined behavior unless player gives any kind of input to them. However, they are the only way to change the existing behavior of the independent objects as well.

1. Locked Door: If the player has a key, it will open, giving access to new room.

2. Light Switch: Turns on/off lights which will only have purely cosmetic value.

3. Button: Triggers some other object (Could be a trap or a door)

4. Pressure Plate: Same as button but it has to be stepped on, or the player can move a block so that the pressure plate stays pressed without having the play standing on it.

5. Lever: Similar to button but unlike the button which trigger an event once, having the lever down will provide constant stream of event (The door stays open / Bridge stays extended / vice versa)

6. Block: Can be pushed around by the player if the player has obtained the prerequisite upgrade.

7. Door: Unlike Locked Doors, these can be opened by triggering mechanisms associated with this door to grant access to other parts of the current room.

* **Terrain**

Terrain types are usually for determining the graphics that a tile will use, but some terrain might have special perks attached to them, the main example being lava, which is still a walkable floor, but it will damage the players when walked on.

1. Floor: Standard terrain type with nothing special about it.

2. Wall: Standard impassable terrain.

3. Bridge: Different graphic version of what is essentially a floor.

4. Lava: Will damage the player when the player is standing on it. Can be rendered harmless with certain item.

5. Light Blocks: Lights up when the light switch is pressed. Purely Cosmetic.

6. Scorched Earth: Behaves exactly the same as lava, but the difference is that it will disappear after certain time.

7. Void: Another version of impassable terrain.

* **Power-ups**

Power ups offer temporary boost to aid in player’s adventure during the play session by altering/modifying player’s stats in a positive way.

1. Restore Health: Restores the health of the player by 1.

2. Burst of Speed (Movement Speed): Increases movement speed by certain amount for certain duration.

3. Invincibility: Player becomes immune to any source of damage for certain duration.

4. Burst of Speed (Attack Speed): Increases attack speed by certain amount for certain duration.

5. Firepower Boost: Increases damage by certain amount for certain duration.

* **Upgrades**

Upgrades, unlike power-ups, offer permanent boost to player’s capability, sometimes unlocking completely new gameplay mechanic with it.

1. Rapid-fire Module: Permanently boosts the player’s shooting speed by certain amount.

2. Quad-direction Bolt Splitter Mk IV: Instead of firing towards the direction that the character is looking at only, player can now fire towards all directions at once, regardless of where the player is looking at.

3. Boots of Speed: Permanently increases movement speed by certain amount.

4. Assault Cuirass: Increases maximum health by 1.

5. Ring of Vitality: Passively regenerates lost life

6. Salamander’s Shroud: Grants immunity to fire based damage (Lava/Scorched Earth/Contact with Magmatrum)

7. Burst Fire Module Mk II: Shoots two bolts instead of one

8. Burst Fire Module Mk III: Shoots three bolts instead of one

9. Gauntlet of Ogre Strength: Allows players to push blocks.

* **Miscellaneous**

Other objects which do not belong to any specific category will belong here, in this case: the Key.

1. Key: The key will spawn when the players satisfy specific conditions which depends from room to room at predetermined spot. If Player picks up the key, the Locked Doors will open.

# 4 Functional Requirements

**1. Play Game**

The purpose of the game is to reach the final room where you will encounter the final boss when defeated, you will win the game. Majority of the player’s stats are prone to permanent upgrade with the items you find during your adventure. The players will need to find keys to gain access to new rooms by completing certain objectives depending on the room type (combat/puzzle/both). Combat involves destroying all enemies in a room which will in turn spawn the key at the center of the room (if applicable), puzzle will involve interacting with certain dependent object which will trigger the spawning of the Key at certain spot.

**2. Change Options**

The user can change two options:

-Texture quality

-Screen size

Texture quality will alter the type of Spritesheet that will be used in the game for tiles/objects/enemies. Screen size is self-explanatory, the system will recommend the player to use high definition textures for larger screen size.

**3. Display Credits**

Simple display of credits for any external sources we might use and list of our names/IDs will be displayed on the screen.

**4. Display Instructions**

As the game does have some degree of depth, the users might want to see full list of controls before jumping into the main game. The system will list them out on the screen.

**5. Open Bestiary**

There will be many different types of enemies and their behavior will be completely unknown for the first time players. So, the Bestiary will contain information about all enemies the players will encounter in the game. They will be listed on the screen by the system.

**6. Pause Game**

Allows players to pause the game in the case they need to take a break, or just tend to other matters.

# 5 Non-functional Requirements

• Game Performance

The game will require to have low latency (100 ms) from each keystroke to make the gameplay as smooth as possible. Object management/rendering will also be a point of concern for performance as late game will offer higher number of objects on the screen simultaneously.

• Graphics Performance

As explained above, late game will feature higher numbers of objects, so rendering them as efficiently as possible is a notable concern to take note of. Also, it would give reason to add texture changing option as this game could be ran on low-end PC or laptop.

• User-friendly Interface

The user must know exactly what is going at all times, which means the UI should not be cluttered with useless information and make it easy for users to figure out what is going on at their screen while they are playing the game.

• Reusability

It is planned to have modular system for level design so that users themselves can create their own content for replayability value. Also, this will reduce the amount of hard coding that would be necessary for level design and allows easy reusal/expanding content for the future.

# 6 Pseudo Requirements

1. The code will be written in Java only

2. Desktop only

3. Level editing should be easy to learn.

# 7 System Models

### 7.1 Use Case Model

**Change settings**

**Primary Actor:** Player

**Interests:** The Player wants to change some settings like texture quality or screen size.

**Pre-condition:** - Player has to be in Main Menu

**Post-condition:** -If player has changed any setting, new settings will be applied on the gameplay

**Entry Condition:** Player clicks on "Change Settings" in the main menu.

**Exit Condition:** Player clicks on ““Return to Main Menu” in the settings screen.

**Successful Scenario Event Flow:**

1. Player wants to change texture quality
2. System returns settings screen
3. Player changes game settings as he wishes
4. System applies the changes
5. Player clicks on "Return to Main Menu"
6. System renders the main menu screen.

**Alternative Flows:**

1. Player wishes to return to main menu
   1. Player clicks on “Return to Main Menu”
   2. System renders the main menu screen.

2. Player wishes to see settings

a. Player gets the settings screen

b. Player checks the settings and doesn't make any change

c. Player clicks on “Return to Main Menu”

d. System renders the main menu screen.

**Display Instructions**

**Primary Actor:** Player

**Interests: -**Players want to see different keybindings and general information on how to play the game

**-**System displays all of these on the screen

**Pre-condition:** - Player has to be in Main Menu

**Post-condition:-**

**Entry Condition:** Player clicks on “Display Instruction” in the main menu.

**Exit Condition:** Player clicks on “Return to Main Menu” in the instruction screen.

**Successful Scenario Event Flow:**

1. Player clicks on “Instructions” in the main menu.
2. System renders all the instruction text on the screen.

**Alternative Flows:**

1. Player wishes to return to main menu
   1. Player clicks on “Return to Main Menu”
   2. System renders the main menu screen.

**Open Bestiary**

**Primary Actor:** Player

**Interests:**

**-**Player wants to get information about enemy types in the game

**-**System displays Bestiary panel on the screen

**Pre-condition:** Player has to be in Main Menu

**Post-condition:**

**Entry Condition:** Player clicks on “Open Bestiary” in the main menu

**Exit Condition:** Player clicks on “Return to the Main Menu” in the Bestiary screen

**Successful Scenario Event Flow:**

1. Player clicks on “Open Bestiary” in the main menu

2. System shows enemy types on the screen

**Alternative Flows:**

Player wishes to return to main menu

* 1. Player clicks on “Return to Main Menu”
  2. System renders the main menu screen.

**Display Credits**

**Primary Actor:** Player

**Interests: -**Player wants to see information about developers of the game and external sources used in game development.

**Pre-condition:** - Player has to be in Main Menu

**Post-condition: -**

**Entry Condition:** Player clicks on “Display Credits” in the main menu.

**Exit Condition:** Player clicks on “Return to Main Menu” in the credits screen.

**Successful Scenario Event Flow:**

1. System displays all the credits text on the screen.

**Alternative Flows:**

1. Player wishes to return to main menu
   1. Player clicks on “Return to Main Menu”
   2. System renders the main menu screen.

**Pause Game**

**Primary Actor:** Player

**Interests: -**The Player wants to pause the game either to 1. Quit the game or 2. Pause it to deal with other issues or look at other things.

**Pre-condition: -**The player is in the game

**Post-condition: -**

**Entry Condition: -**The player presses the corresponding key that opens the Pause Menu

**Exit Condition: -**The player presses “Resume Game” OR “Quit Game”

**Successful Scenario Event Flow:**

1. The system shows the Pause menu.

**Alternative Flows:**

1A. Player wishes to return to main menu

1. Player clicks on “Quit Game”
2. System renders the main menu screen.

1B. Player wishes to continue the game

* 1. Player clicks on “Resume Game”
  2. System resumes the main game loop.

**Play Game**

**Primary Actor:** Player

**Interests:-**The Player aims to beat the game by defeating the final boss.

**Pre-condition:-**The system will use default settings unless the player has changed the settings beforehand.

- Player has to be in Main Menu

**Post-condition:-**

**Entry Condition:** Player clicks on “Start Game” in the main menu.

**Exit Condition: -**Player clicks on “Quit Game” in the pause menu.

-Player defeats the final boss

-Player loses all life

**Successful Scenario Event Flow:**

1. Game is initialized by the system.
2. Player starts from the first room.
3. Player explores the room.
4. Player finds the key.
5. System unlocks the door allowing player to access new room.

*Repeat step 3 to 5 until player slays the final boss or loses all health*

1. System displays congratulation screen and returns to main menu.

**Alternative Flows:**

3A. Player wants to pause the game.

1. Player opens pause menu
   1. Player clicks on “Quit Game” to return to main menu
   2. Player clicks on “Resume Game” to exit from pause menu and continue the game.

3B. Player picks up an item

* 1. If Player picked up a power-up, boost the player’s corresponding stat according to the power up temporarily.
  2. If Player picked up an upgrade, boost the player’s corresponding stat permanently according to the upgrade picked up.

3C. Player collides with the Enemy or any kind of damaging element.

a. The System recalculates the Player’s health

a. If the Player has 0 life after recalculation

i. The system displays “You have died” and sends the user back to the main menu.

b. If the Player has more than 0 life after recalculation

i. Continue the game.

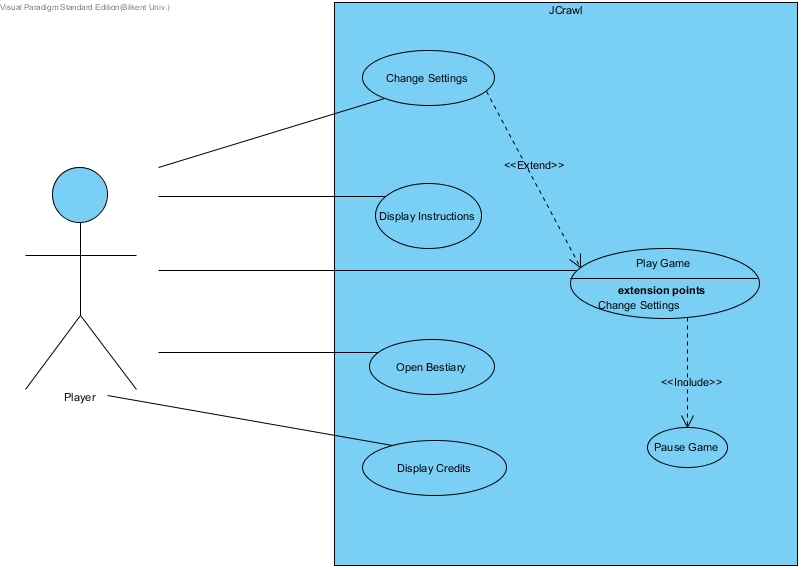
3D. Player completes a puzzle or kills all enemies

a. The system spawns the key at the predetermined spot.

3E. Player defeats the final boss

a. Move to successful event flow no. 6

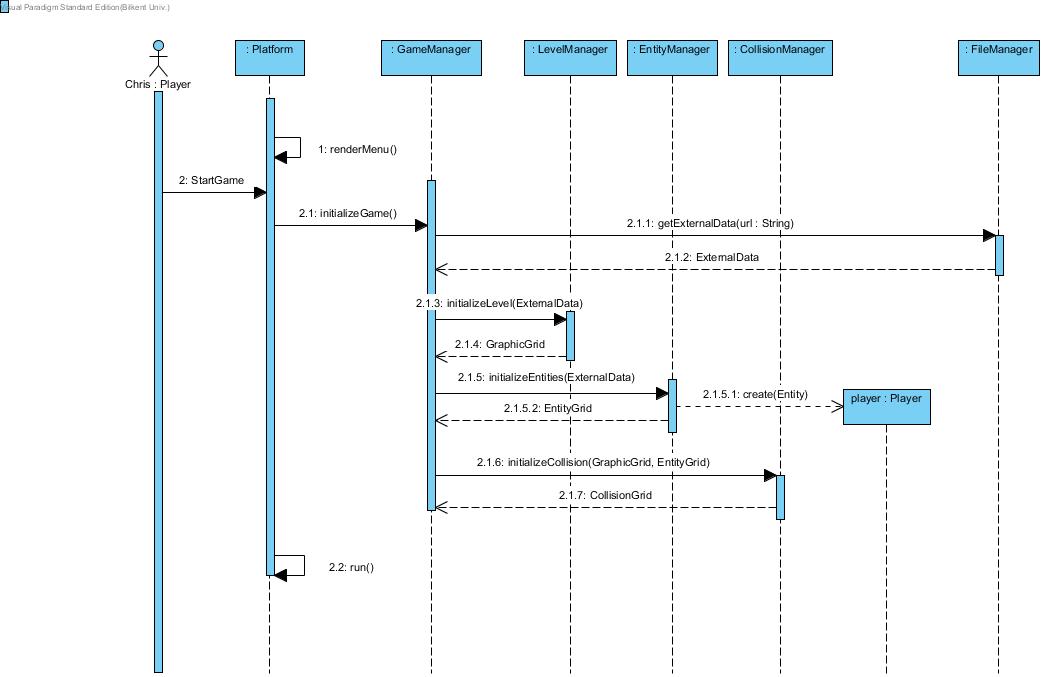
Our use case model can be best summarized by the following use case diagram show below. (Figure 1)



**Figure 1 – Use Case Diagram**

### 7.2 Dynamic Models

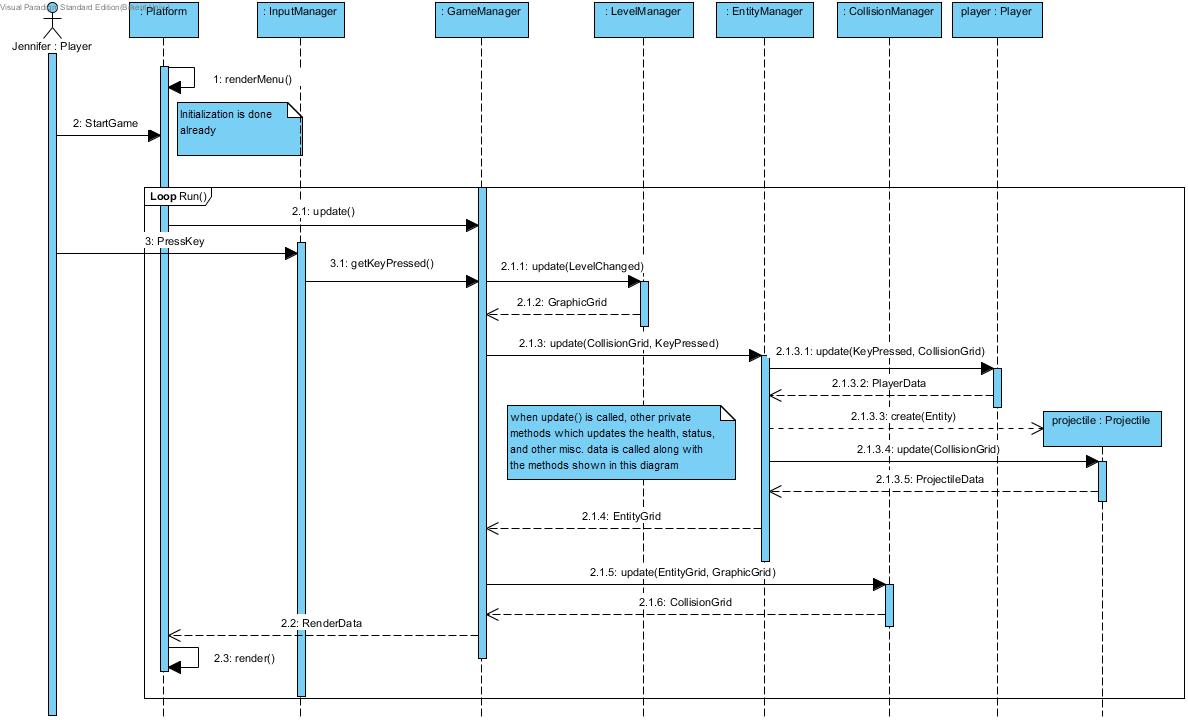
1. **Scenario 1:** After Chris started the game, the system initialize graphics and objects according to the file of Chris. He forgot to place any other entities except for the player, so only the player was rendered and there was no other entities he could interact with.



**Figure 2.1**

Here, the Platform renders the menu of the game using RenderMenu() method, then the player presses on the “start game” which causes the Platform to call the InitializeGame() method, which in turn calls the related initialization methods of all relevant managers (InitializeLevel(), InitializeEntities()…) All these initializations return their corresponding data which in turn will be used by the Platform later on to render the screen (which is not shown here yet).

1. **Scenario 2:** Jennifer started the game, the system loaded first level with blocks, independent and dependent object and enemies, she decided to test some of the functionalities of the game before moving onward to the next room, she fired some projectiles at the wall.

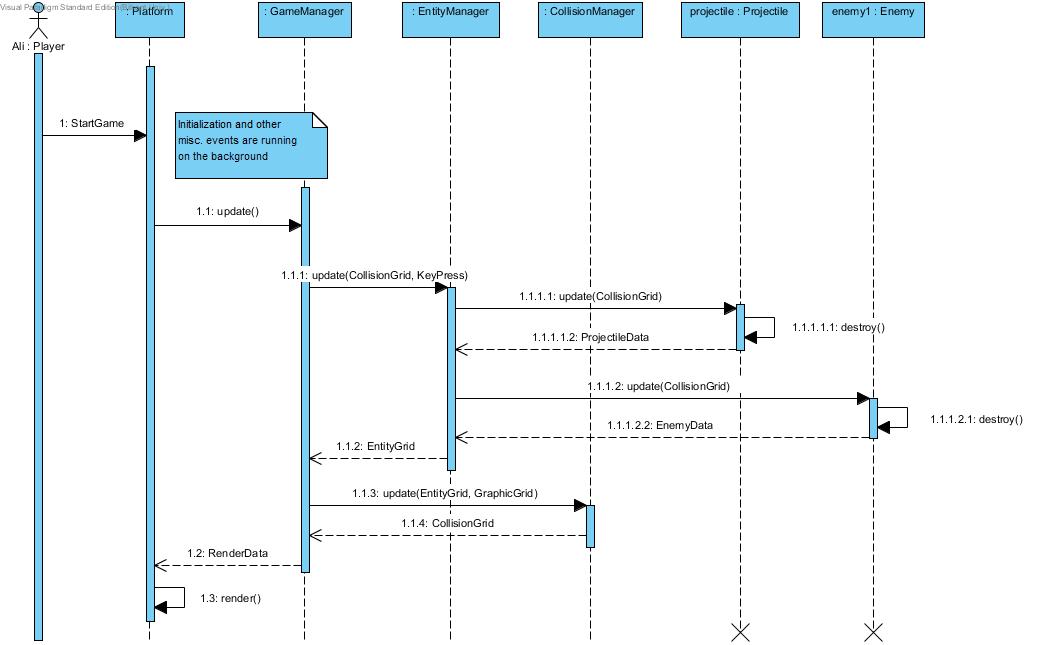


**Figure 2.2**

Here, the actual main game loop is shown (although not all possible iteration of it), the main game loop will be in the method Run(), and divided largely into two parts: Update() and Render(). Update method will go through every existing gameplay elements and update their position, health, status and other misc. data associated to them every tick (preferably 60 updates per second, aka. UPS) After data update is complete, these data will be parsed as data which the platform can read and actually render it on to the screen by GameManager.

Also, creation of new gameplay entity is shown here; projectile. Through collection of private methods, the EntityManager uses all the necessary information given to it and decides the exact location, direction of its trajectory.

1. **Scenario 3:** Ali performed a test in the custom level. He created a Slime, he shot some projectiles into the Slime and he destroyed Slime.



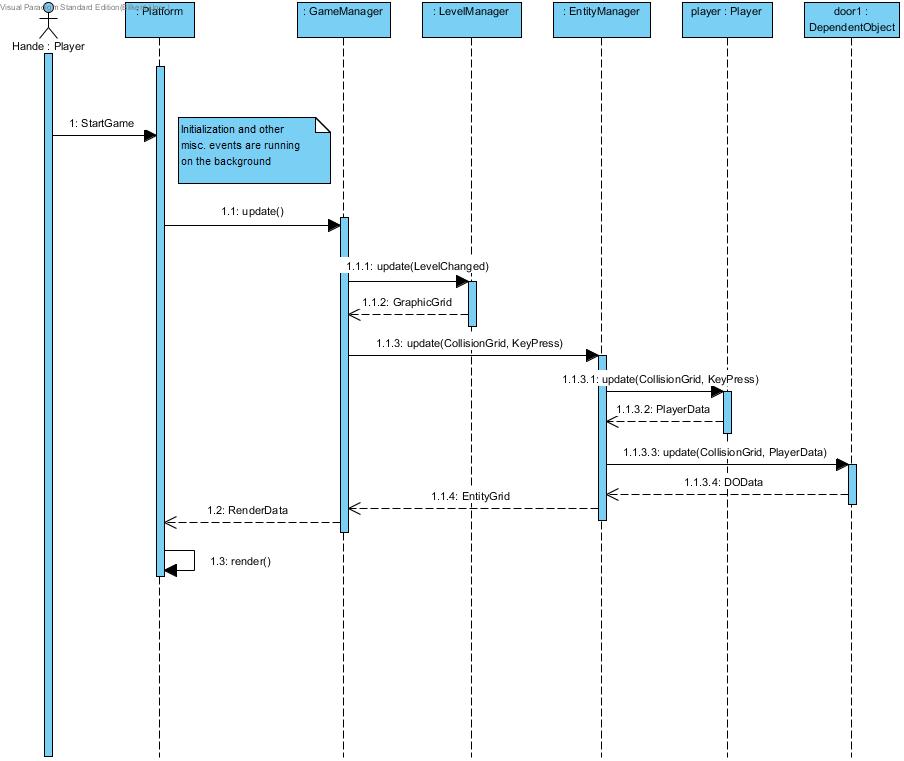
**Figure 2.3**

This diagram showcases some of the internal methods for the Entity abstract class, Destroy(), since all objects have to be destroyed when the level changes, all entity objects will have this method. The current scenario is where the projectile collides with the enemy, reducing its health point to zero, thus invoking the Destroy() method, removing it completely from the game, more specifically, EntityManager.

This interaction occurs during the update() method and within EntityManager.

CollisionGrid and EntityGrid are data storage classes which is basically equivalent to the struct in C++.

1. **Scenario 4:** Hande already took care of multiple enemies which was roaming the room, and she already picked up the Key from the center of the room. She then proceeded to the Locked Door and because she had the key, the door opened and she was able to proceed to the next room.

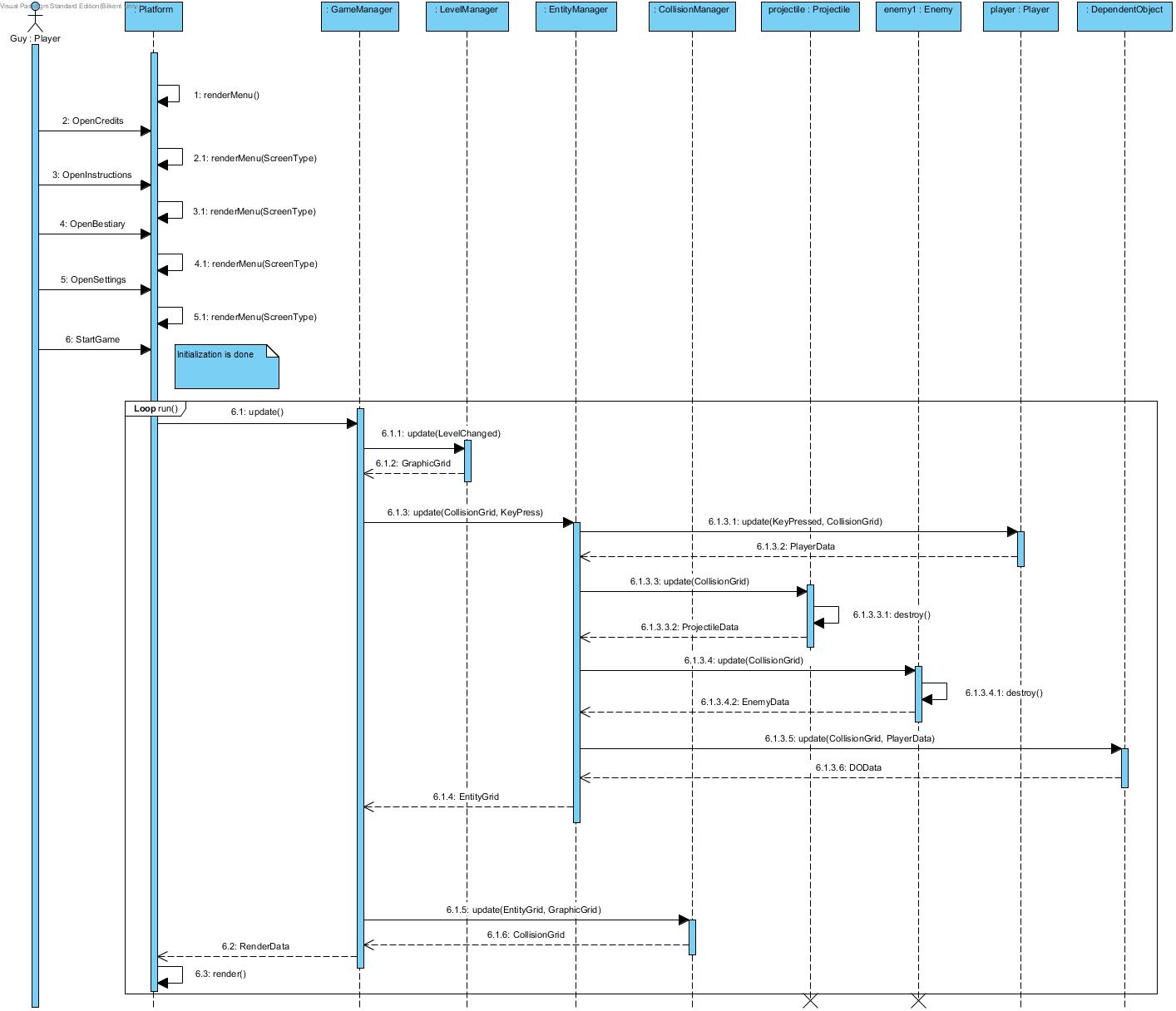


**Figure 2.4**

This scenario showcases interaction between Player and Dependent Objects, more specifically, the door. CollisionGrid update is omitted here since it is always executed in a loop, however the LevelManager update() is included as the interaction between the door and the player results in change of level data (GraphicGrid).

Again, all of this interaction/data manipulation occurs during update() and within EntityManager.

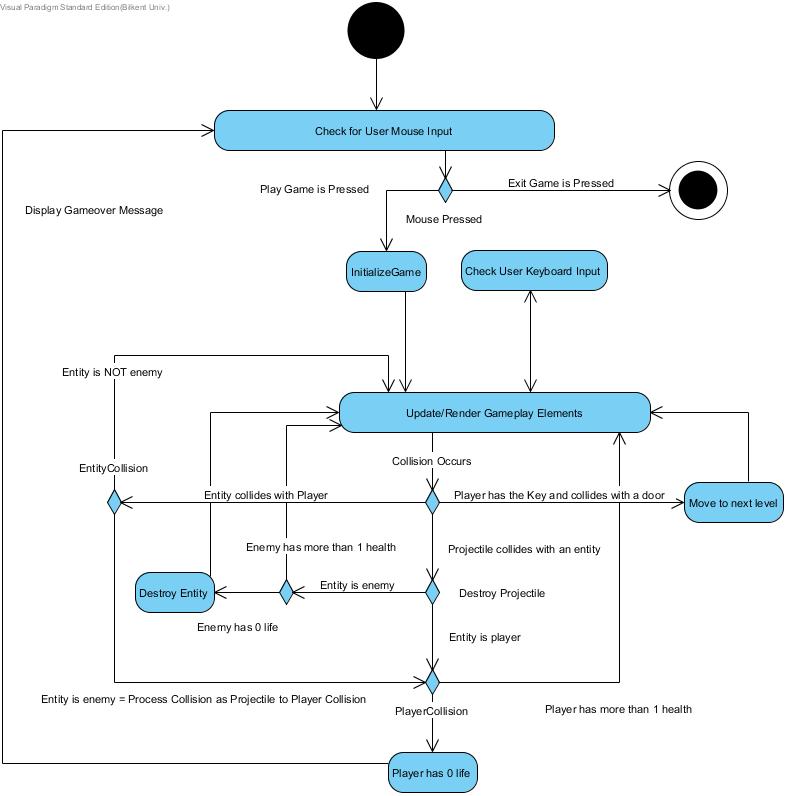
1. **Scenario 5:** Guy starts up the game, and he looks through credits, instructions, bestiary and changes the texture pack. After messing around with every menu option he can find, he starts the actual game, wait for the initialization to finish. After the game loaded, he proceeds to shoot at some enemies and after few salvos, he manages to kill them. After he killed them all, the key was dropped at the center of the map. He picked up the key and he picked up an upgrade along the way, boosting his movement speed. He moved to the next room. Unfortunately, he forgot to design multiple rooms so the next room was the last room, so he won the game after finishing that room as well, returning to the main menu.



**Figure 2.5**

Here, the combination of all sequence diagrams is displayed, all options except StartGame player have chosen in the main menu do not require any other class except Platform, Platform uses RenderMenu method to display these options. Other objects are called when actual game is initialized. When the game is started, the system will be in loop which is the Run() method. This loop is used in order to keep the game updated/rendered. Firstly, GameManager updates LevelManager, EntityManager and CollisionManager in order, and gets them data which is required for updating another manager. In this sequence, EntityManager is shown more detailed, EntityManager updates player, enemies, etc. and returns combined data to the GameManager. At the last step, GameManager returns the data to the Platform to display new situation on the screen. Dependent Object is a child class of Entity which is a data storage class.

The following diagram is the Activity diagram of the use case: Play Game.

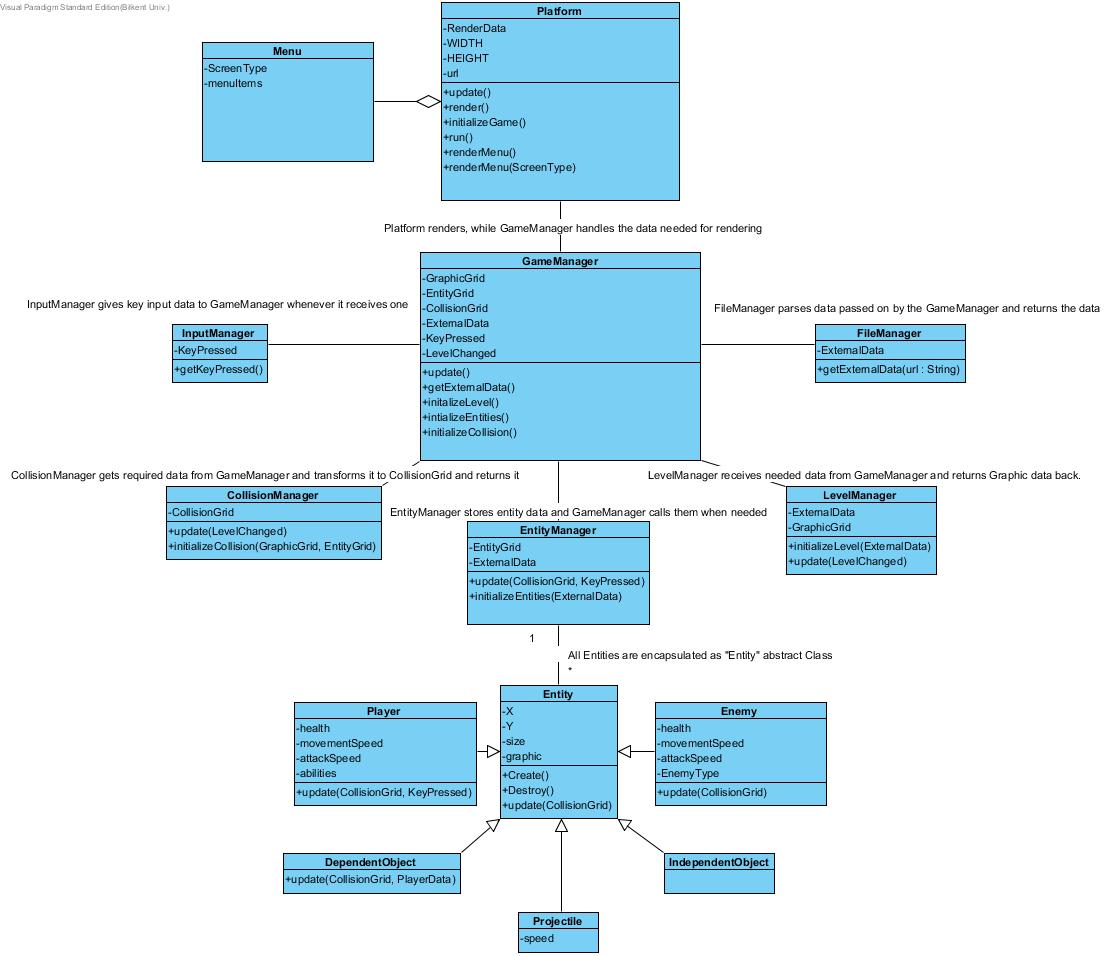


**Figure 3**

When the game first starts, it will first render the main menu where the player can select multiple buttons for different events, but for this diagram, it will cover the “Play Game” as it is the most important and complex event of this software. When the user starts the game by pressing the “Play Game”, all managers go through initialization. When it is finished, the game will be fully rendered on the screen and will be ready to take user input (Key presses) and the system will update/render gameplay elements accordingly.

One primary action that the system will perform will be determining collision and its after-effects. If the collision occurs, first it will check whether the collision occurred between the player and the door, or player collided with a different entity, or an entity collided with a projectile. For the first case, if the player has the key the level will change, and all other entities except player will be destroyed, player will be set to initial start position of the level and new entities that belongs to the level created. If the player doesn’t have the key, nothing will happen. For second case, if the player collided with an entity, it will first check whether it’s an enemy or other object. If it’s not an enemy type object, it will perform its predetermined update method. If the entity was an enemy type, the collision will be handled as if it was like (not exactly) projectile to player (or entity) collision. For third and last case, it will destroy the projectile and reduce the health of the player or entity it has collided to, if the entity has higher than one health point after calculation. If it doesn’t, however, if the entity is a player and the player’s health point goes below 1 after calculation, the game will be over and the user will be forced back to the main menu after rendering game over screen, if the entity is an enemy, it will destroy the entity from the game and the game will continue.

### 7.3 Object and Class Models



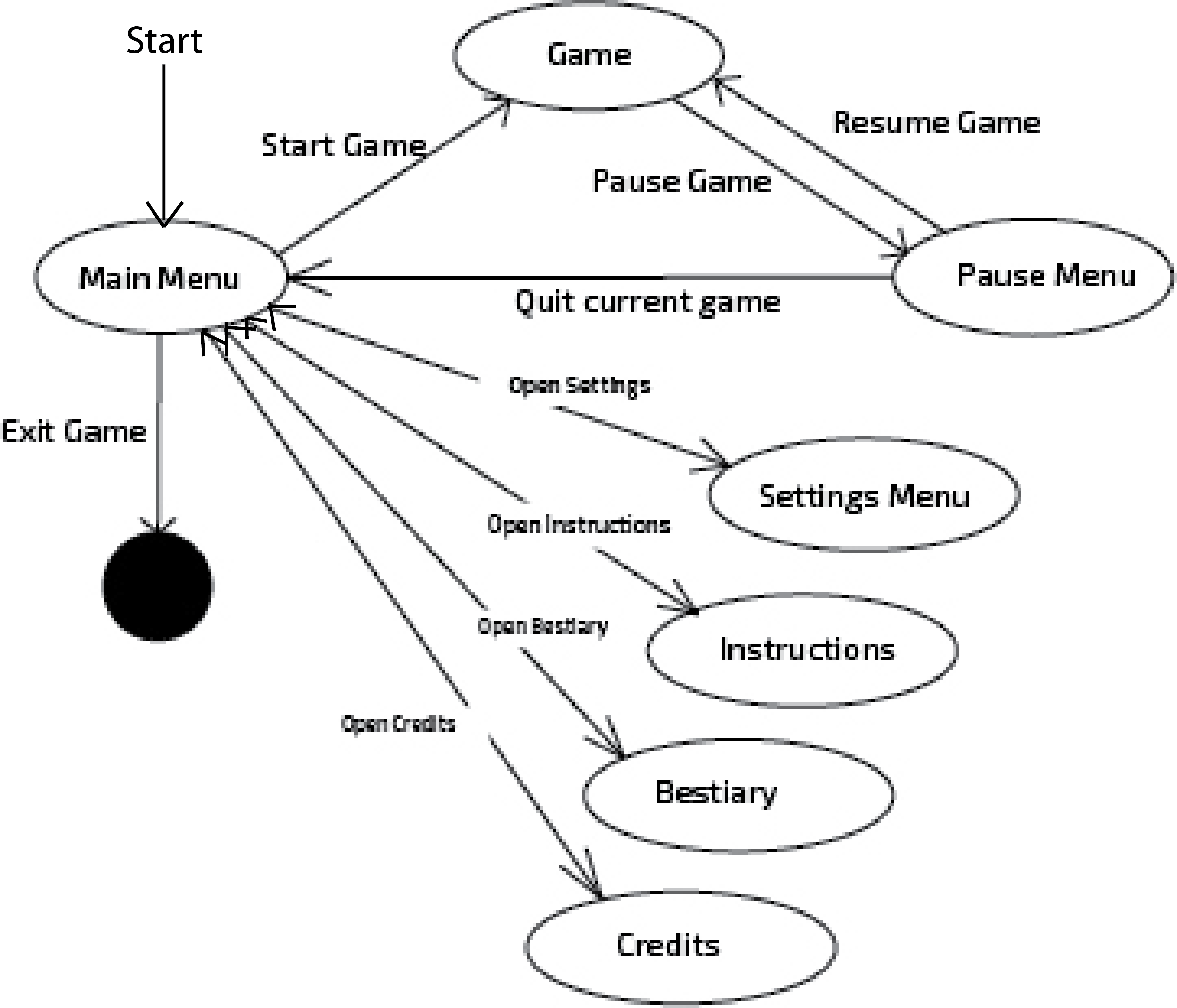
**Figure 4**

In our architecture, the program will be largely divided into “Graphics” section and “Data” Sections, where the managers are purely handling only data, while the Platform and Menu class is in charge of rendering the data into actual graphical objects visible on the screen. (Although there are more components to it, it is abstracted out in this diagram)

GameManager acts as a data bus for all other managers to parse their data into RenderData which the platform class can work with, while each managers manage the data type associated to their names (e.g. CollisionManager manages Collision, and vice versa)

All Gameplay objects except background elements will be under the parent class “Entity”, which will be used to make data movement to other managers easier and faster.

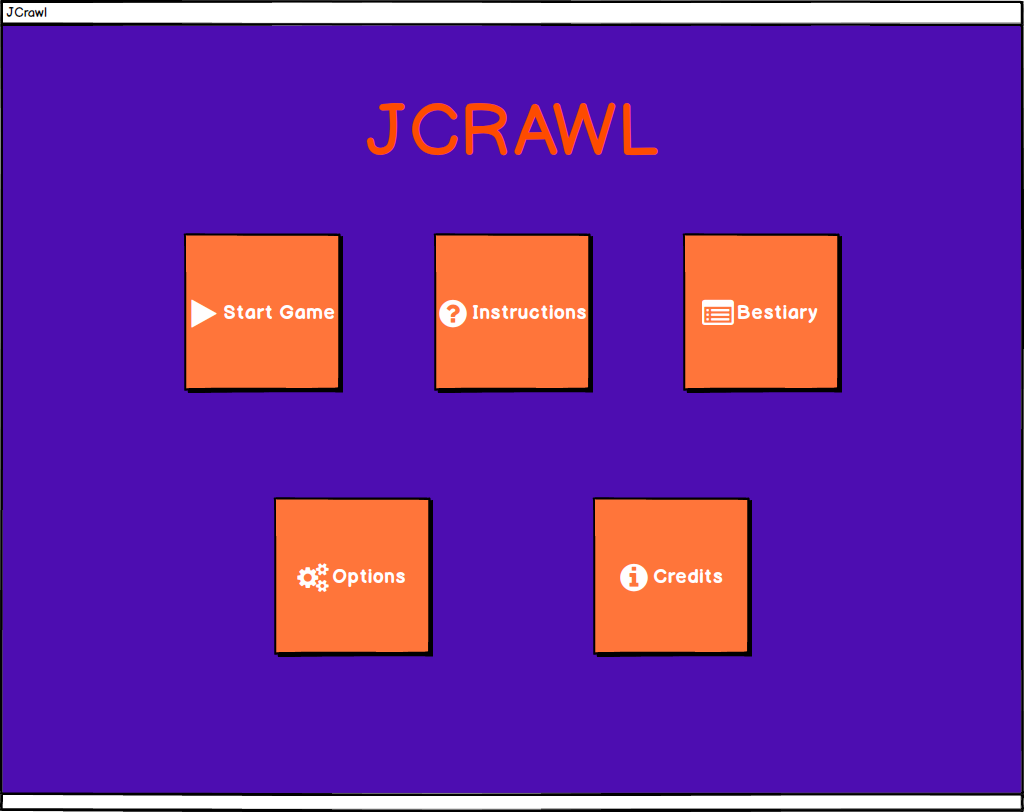
### 7.4 User Interface – Navigational Paths and Mock up Screens



**Figure 5**

All graphics designed by Fatih Taş

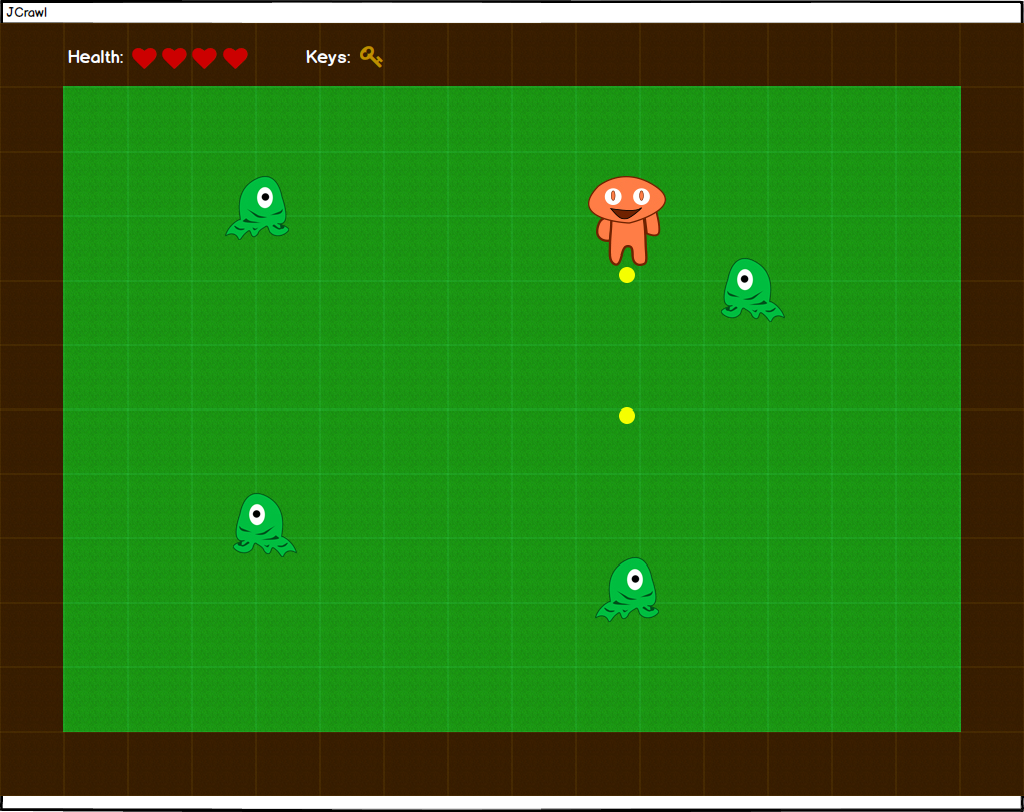
* **Main Menu**



**Figure 6.1**

The first screen that player will see when application start is main menu. There are several options that player can select which are starting game, displaying instructions, displaying bestiary, displaying options, displaying credits. For returning to main menu after selecting one of these options player can click on the return to main menu button at top left of the screen (except play game screen).

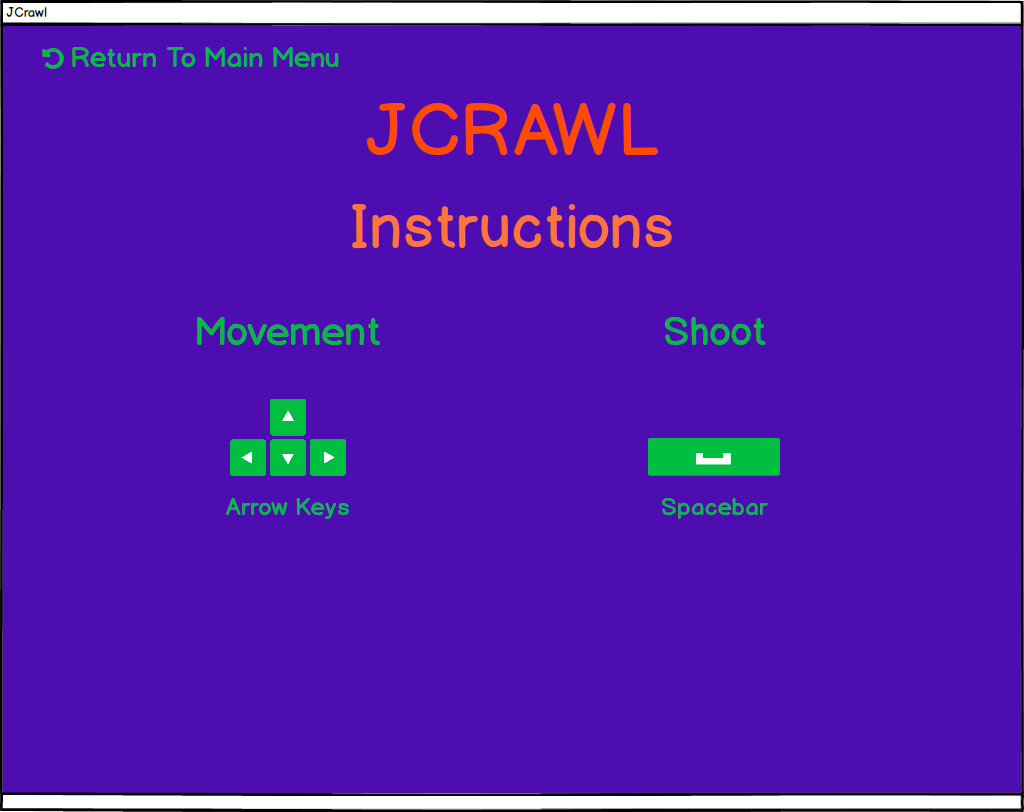
* **Play Game**



**Figure 6.2**

When player selects start game option on the main menu, system will initialize game objects and render them to screen. On top left of the screen player’s health and keys are displayed.

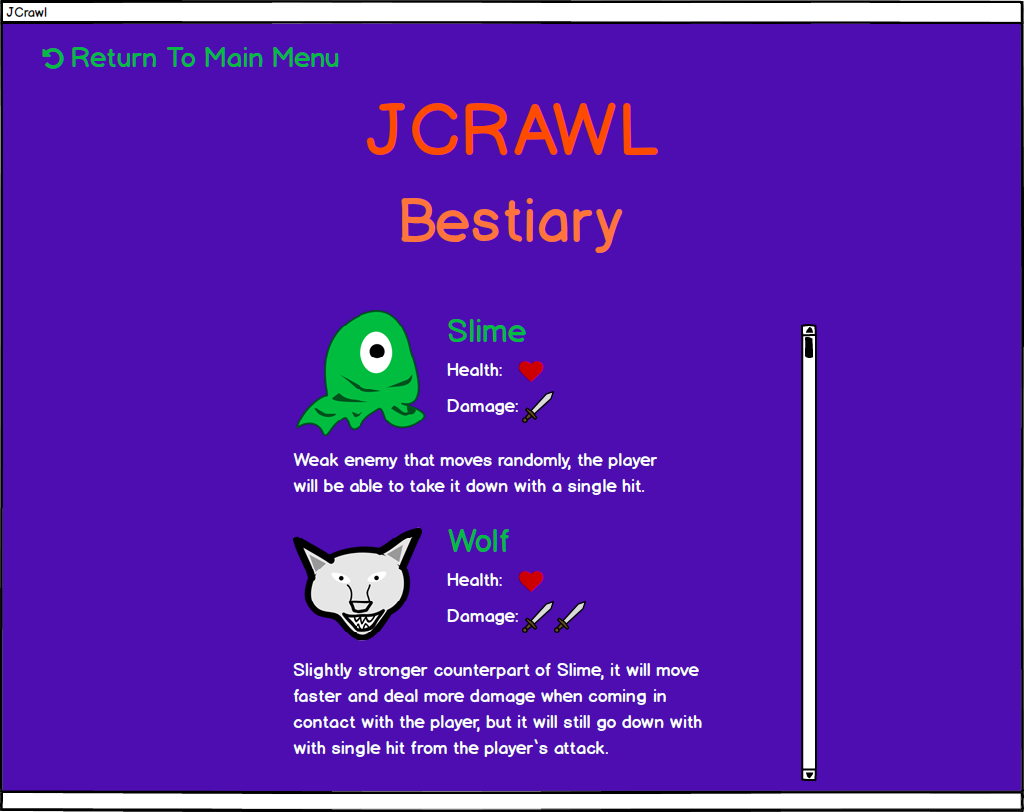
* **Open Instructions**



**Figure 6.3**

When player selects instructions on main menu, instructions screen will be displayed. On instructions screen keyboard shortcuts for moving the player and shooting projectile are displayed.

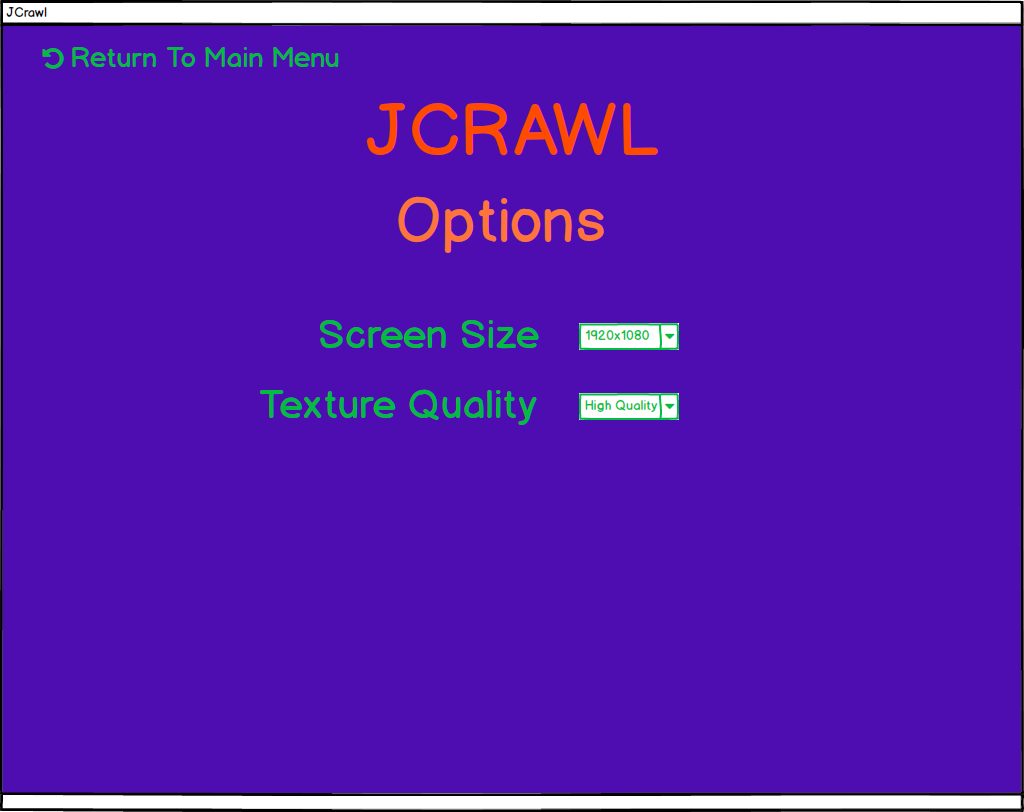
* **Open Bestiary**



**Figure 6.4**

If player clicks bestiary button on main menu, bestiary screen will be shown. On bestiary screen, information about enemies (health, damage, behavior vice versa) is displayed.

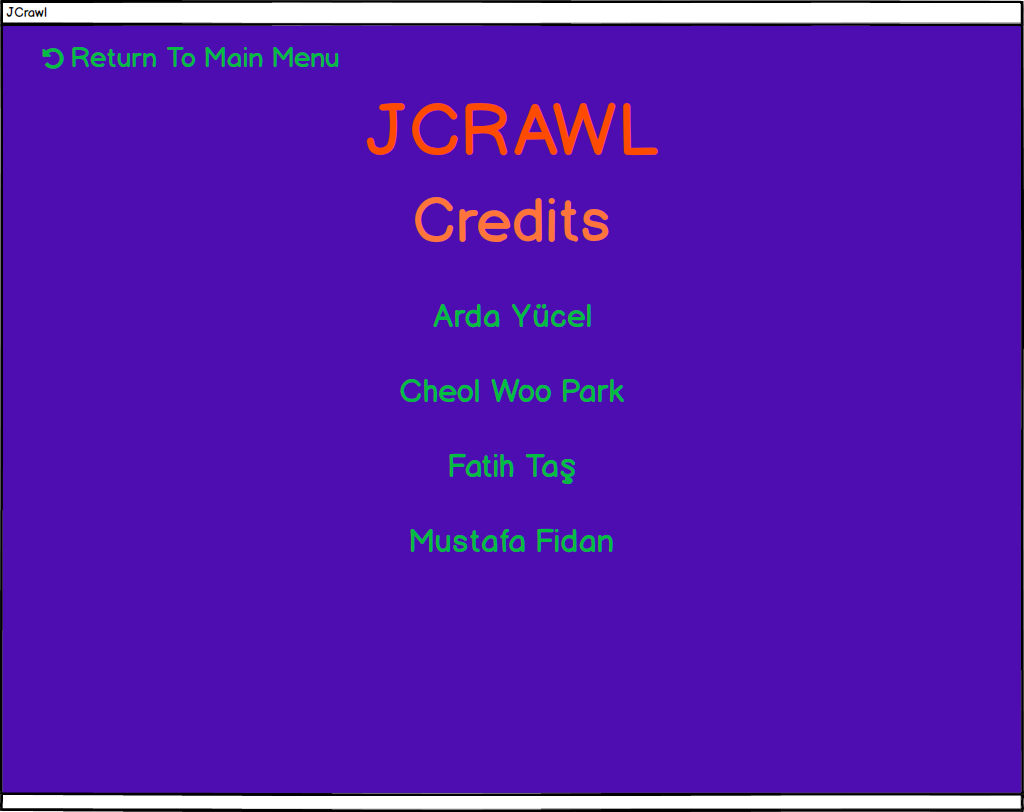
* **Open Settings**



**Figure 6.5**

On options screen, player can change screen size and texture quality by selecting one of the items in dropdown menu. If player doesn’t change these settings, system will use default settings.

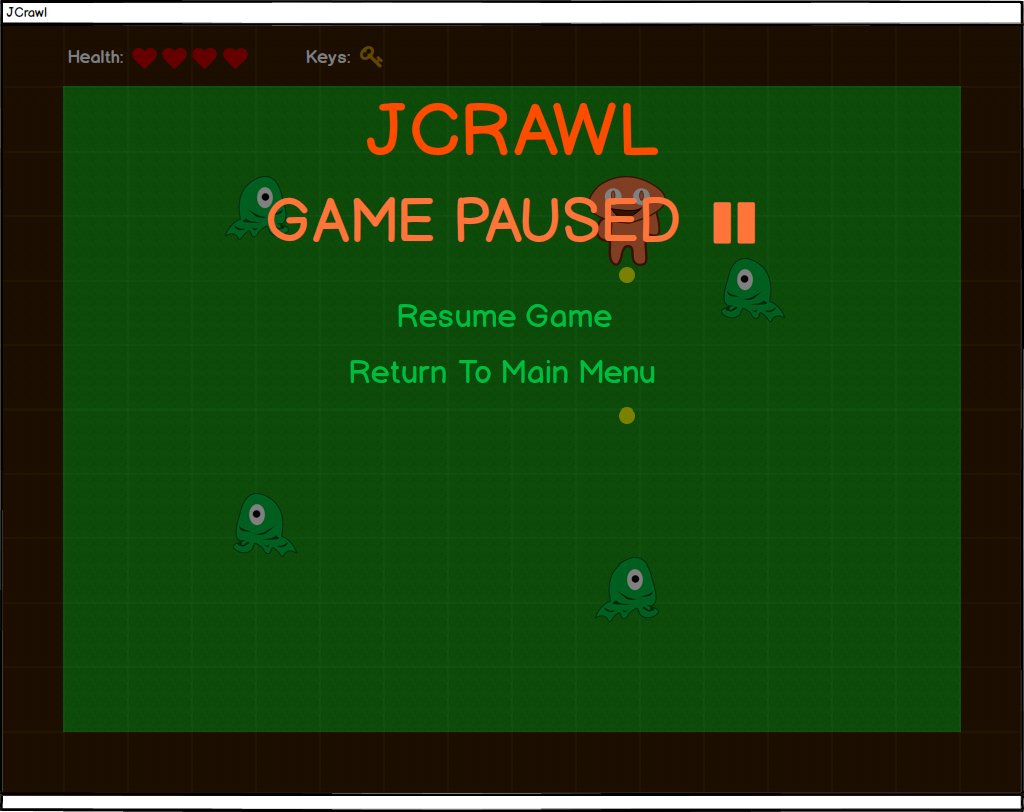
* **Open Credits**



**Figure 6.6**

When player selects credits option on main menu, credits screen will be displayed. On credits screen, names of the developers are displayed.

* **Pause Game**



**Figure 6.7**

When player pauses the game, pause game menu will be displayed. On this menu there are two options which are resume game and return to main menu. Player can continue playing by clicking resume game button.

# 8 Glossary

Melee: Another word for close quarters.

Spawn: Create, generate.

# 9 References

Object-Oriented Software Engineering, Using UML, Patterns, and Java, 2nd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2004, ISBN: 0-13-047110-0.