BILKENT UNIVERSITY
ENGINEERING FACULTY
DEPARTMENT OF COMPUTER ENGINEERING

CS 319
Object Oriented Software Engineering Project
e-Bola

Final Report
Group 2B

Can AVCI
Deniz SİPAHİOĞLU
Ergün Batuhan KAYNAK
Esra Nur AYAZ

Course Instructor: Bora Güngören

October 4, 2017

Table of Contents

1. Introduction

In this report, it will be explained if the expectations from the project were fulfilled after the implementation. There were lots of features explained in both the Design and Analysis reports, and some of these features are included in the current version of the project and some of them are not. There are different reasons for our choices of implementation, including memory usage, graphics quality and saving/loading the game.

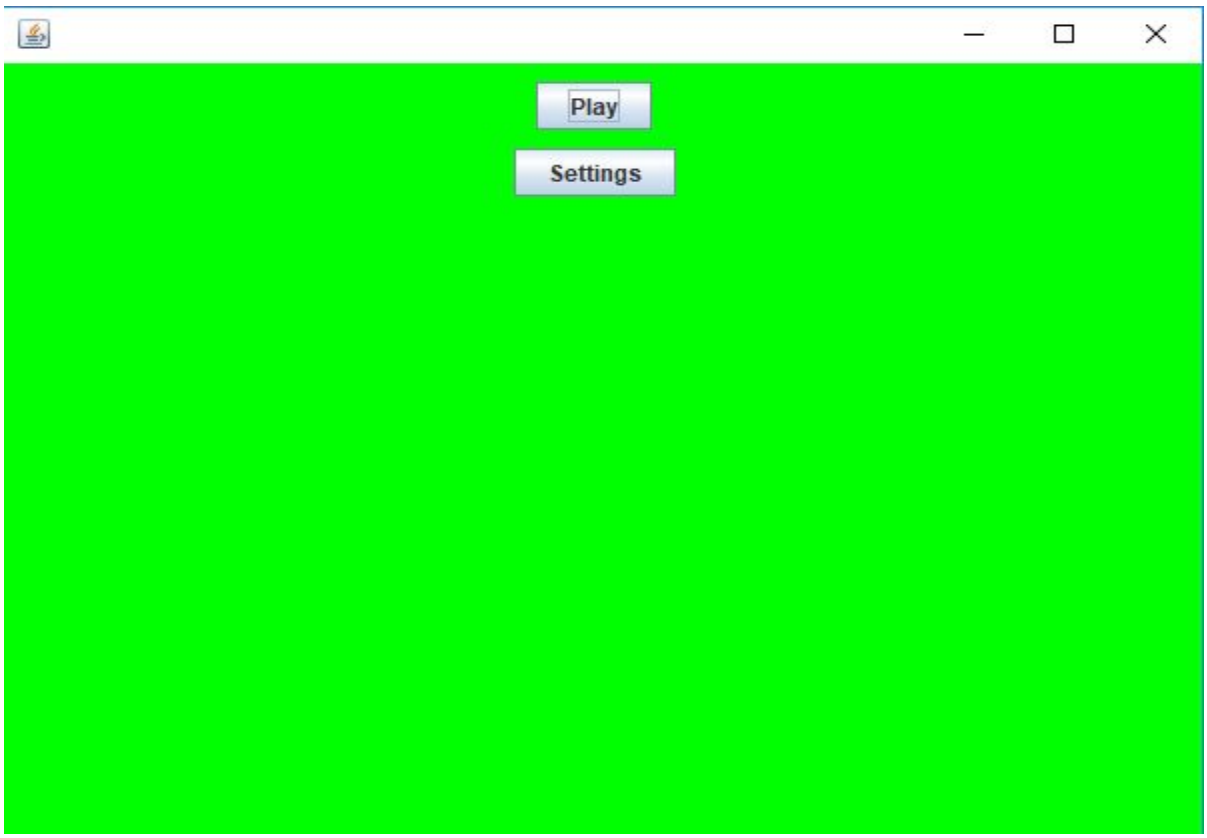
2. State of the implementation

Right now, user is able to navigate in a menu with limited options. User can click the "Play Game" button to start the game. When this event occurs, game reads the room data of the first level of the game from the related text files, loads the required resources located within the game files and creates the game. The user can move within the map using arrow keys. We believe that it will be easier to add our features when the engine is at a reasonable stage, so most of the implementation in this iteration focused on the main skeleton of the game engine.

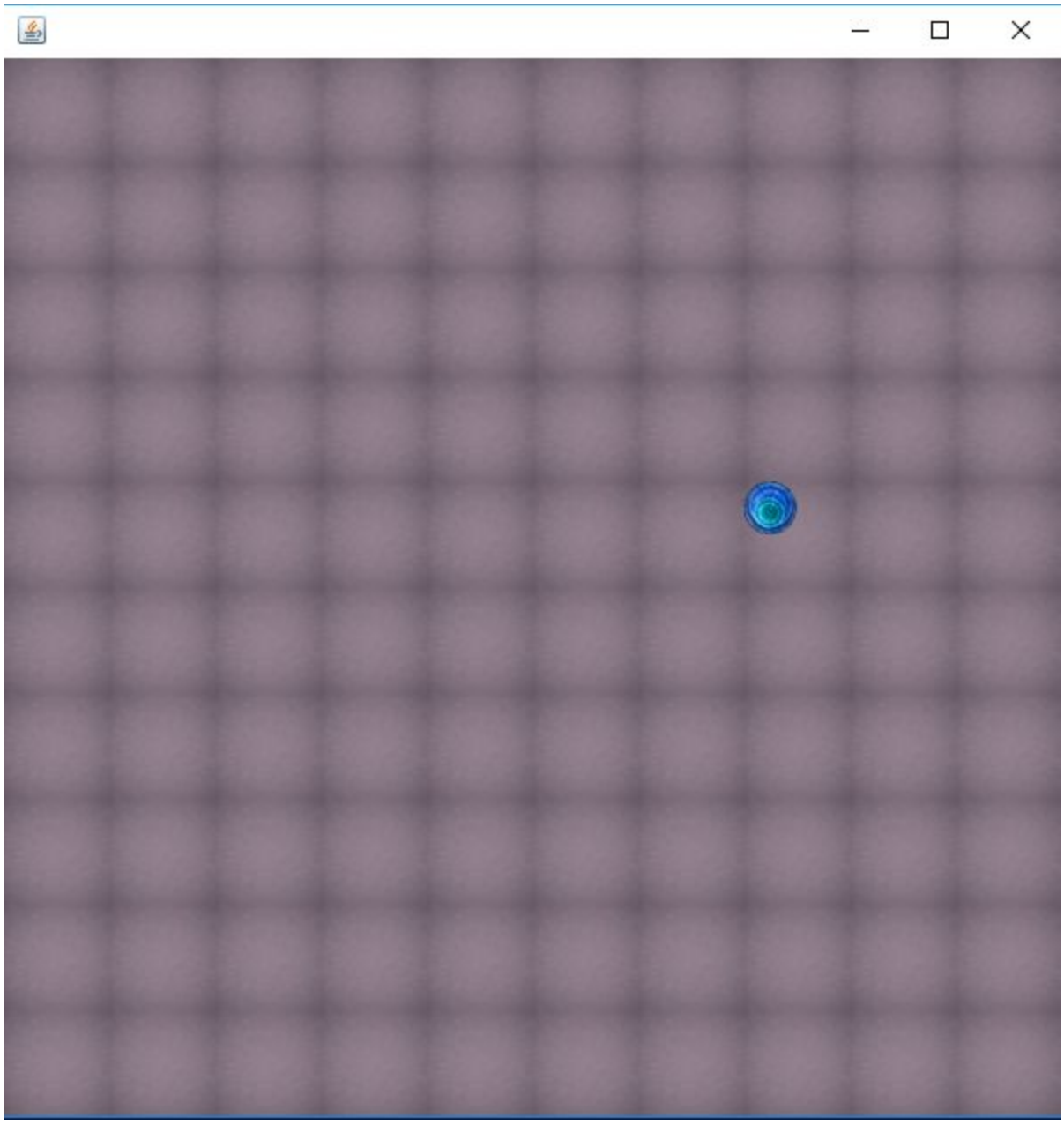
The code of e-Bola consists of 24 classes. These classes are members of the most fundamental packages. The following packages are partially implemented:

- User Interface Manager
- Game Engine
- Input Management
- Map Management
- Reader
- Entity Management
- Game Entities

Screenshot of the main menu:



Screenshot of the game panel:



3. Changes made during the implementation

During our implementation, we saw that some of our design decisions created problems that we have not anticipated during the design report. Some of them were changed as they created big problems in our implementation. They are listed here:

- We noted that we would use wave audio files in order to play the songs, but we realized that using this type increased our memory requirements drastically (by ~500 MB), so we decided to change the format of the audio to mp3.
- Printing both images of static entities (e.g. children of **Tile** class) and dynamic entities(e.g. children of **Alive** class) to the game panel proved to be not possible with Java's **JPanel** class. So, we had to switch our **GamePanel** class to use **JLayeredPane** to paint our game objects. This resulted in adding 3 more classes to our User Interface Manager package: Layer, DynamicPanel, StaticPanel. Layer represents a layer in JLayeredPane. DynamicPanel and StaticPanel extend Layer and are responsible of painting dynamic and static entity objects respectively.
- Due to some shortcomings of Java libraries, having only **int:movementSpeed** for **Celly** class proved not to be enough. Instead, the differences in the movement speed of Celly caused by children of the **Tile** class will be reflected using **int:movementSpeedModifier** attribute. Also, the movement speed is now represented using **int:velocityX** and **int:velocityY** attributes, denoting movement speed in x-axis and y-axis respectively.
- **InputManager** class uses a new interface, InputListener, to communicate with the **GameEngine** class. **InputManager** does not use **Controls** enum as it was designed to in the design report. (See section 4 for more information)

- Instead of delegating all decision making and status checks to **EntityManager** class, **CellyManager** now does more work before the information is sent to **EntityManger** for further actions. Functions such as **void:evaluateInput** and **boolean:checkIValidCoords** are current examples of this change.

4. Planned changes

During our implementation, we also saw that some of our design decisions proved not to be as fruitful as we had hoped. They even elongated the development time. We plan to revert some of these decision for the second iteration if they continue to not prove to be useful. Mainly, they are:

- **InputManager** class proved to be too trivial, and trying to implement it as distinct functionality from the **GameEngine** class gave birth to additional classes for communication.
- **Point** class was going to be used to transfer location info of **Alive** entities in a more compact form. This class proved to be too trivial. Overhead caused by packing and unpacking of the object increased complexity of the code.

There were also some issues that caught our attention when we wanted to implement them. These will be implemented after their design is set. Some of the difficulties we faced are:

- We wanted to play background music throughout the level and make sounds while the **Alive** entities are fighting. We realised that this requires threading operations and might require a more throughout design.

- We get our room data from text files. **Celly** moves between rooms (**Room** objects) by using Portals (**Portal** objects). Binding of these portals to their destination rooms is a crucial part of our game. Our initial reading algorithm proved to have bugs. The algorithm will be reconsidered.
- Right now, the game has scaling problems when the game window is resized. We hope to work towards a solution.

5. Incomplete Parts

- We only have a couple of implemented classes in **GameEntities** package. Implementing them will be easier with the **GameEngine** state we achieved during first implementation cycle.
- Reader package does not have all its classes become some features that require them are not yet implemented. (e.g. **SoundReader** class)
- Enemy AI is not yet implemented.

6. Installation Guide

Implementation is done with Java programming language. Users need to download and install Java Runtime Environment (JRE) compatible with their operating system to play the game.

The game is yet to have jar support. To make the game work, the following steps should be sufficient (instructions for Windows 10):

1. Download the project as a zip file from https://github.com/ebatuhankaynak/2B.E-bola
2. Extract zip file and navigate to the “FieldReady” file

3. run “EbolaGame.bat”

If the user has Java Development Kit (JDK), these steps can be taken also:

1. Download the project as a zip file from https://github.com/ebatuhankaynak/2B.E-bola
2. Extract zip file and open powershell / command line
3. Use `javac EbolaGame.java; java EbolaGame` for powershell or

   `Javac EbolaGame.java && java EbolaGame` for cmd.exe