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

December 16, 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

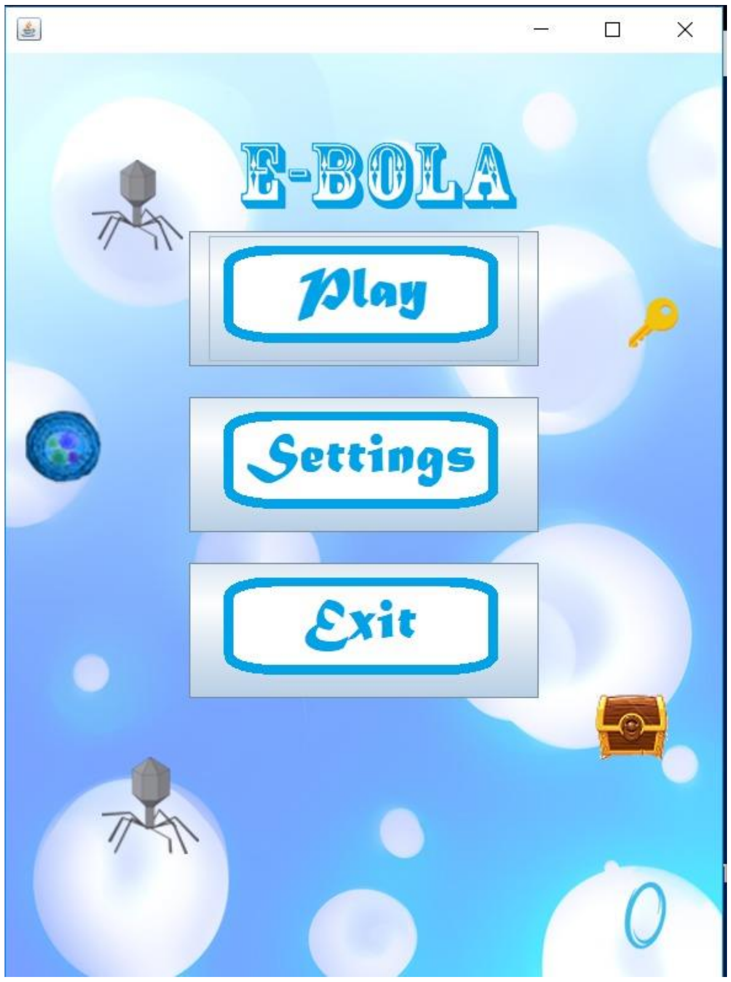
Right now, the user is able to navigate in a menu with limited options. User can click the “Play” button to start the game, “Settings” button to change the audio settings in the game, and the “Exit” button to exit the game. When the “Play” event occurs, game reads the room data of the first level of 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.

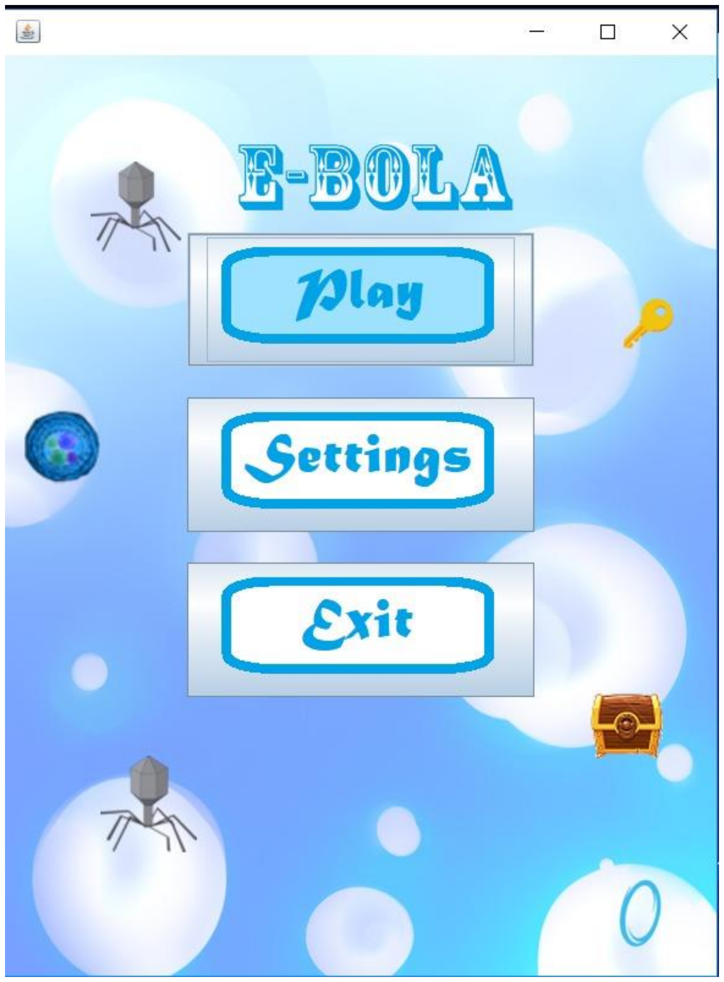
The code of e-Bola consists of 40 classes. These classes are members of the most fundamental packages. The following packages are implemented, but their functionality is limited:

- User Interface Manager

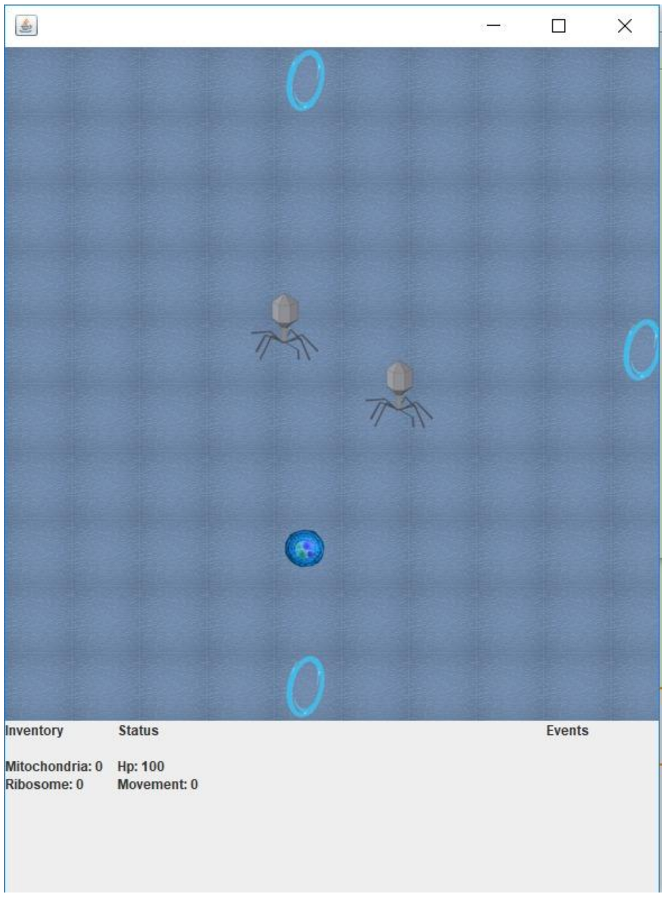
- Game Engine
- Input Management
- Map Management
- Reader
- Entity Management
- Game Entities
- Sound Manager

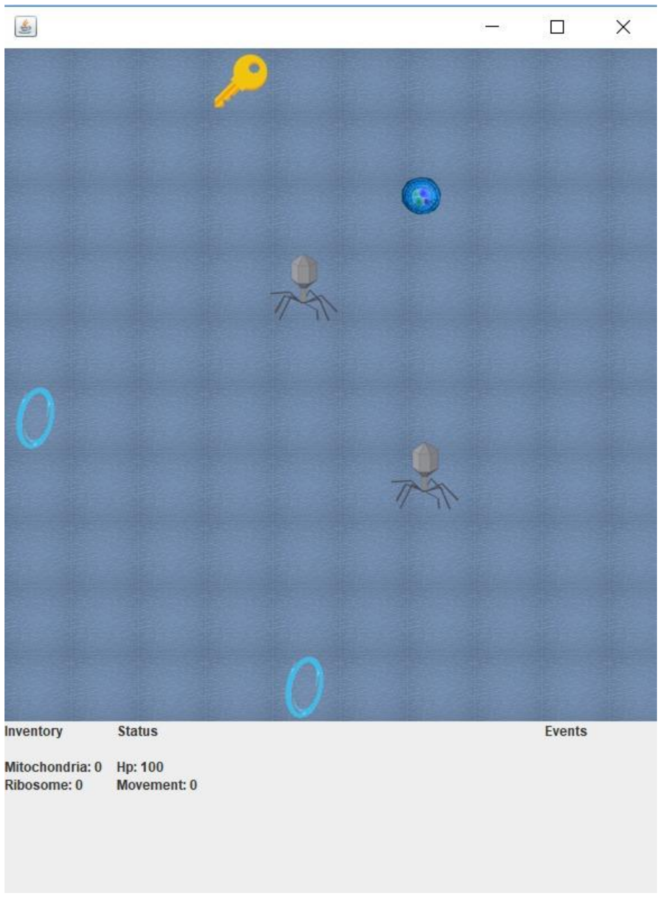
Screenshot of the main menu:

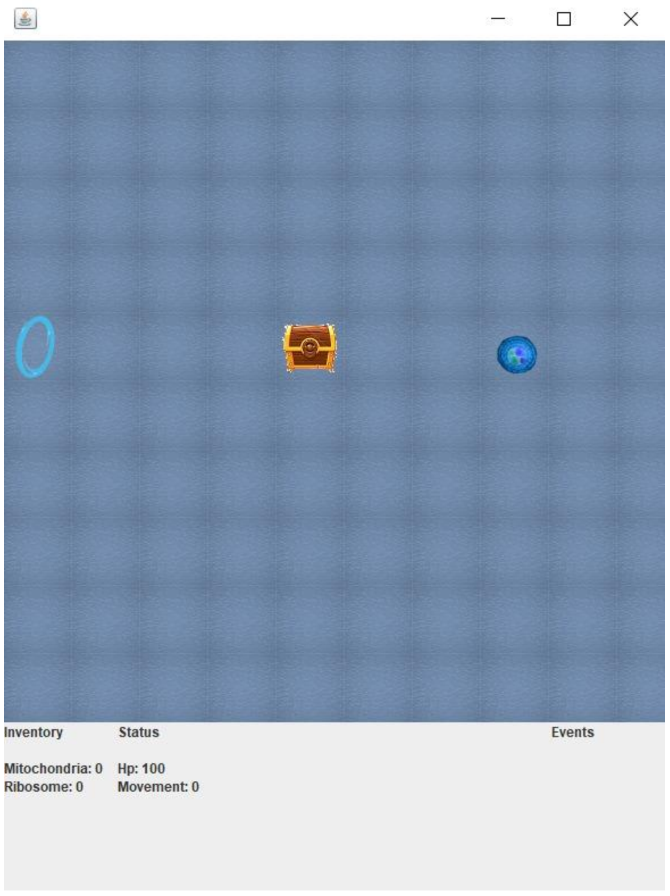


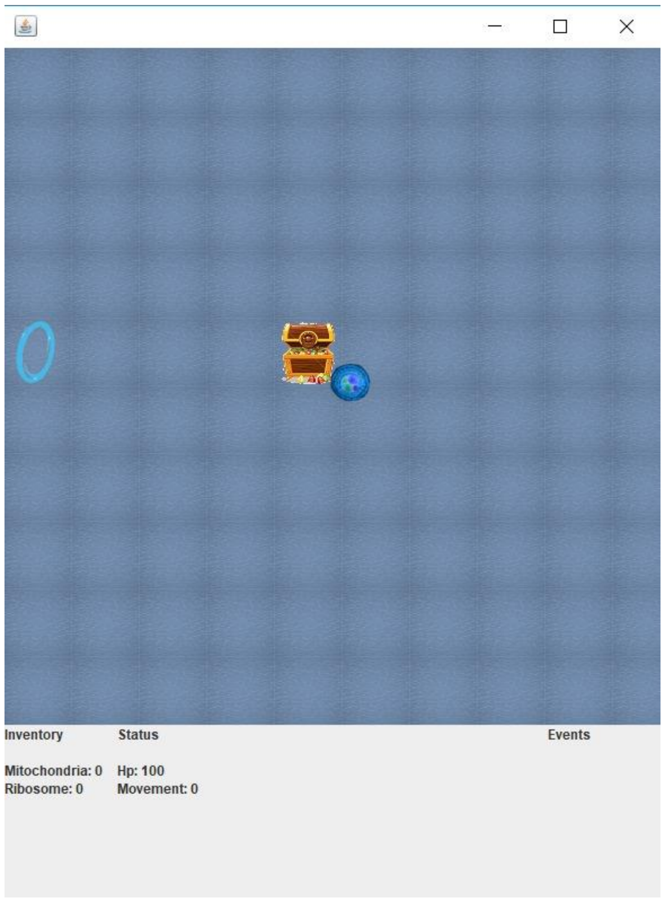


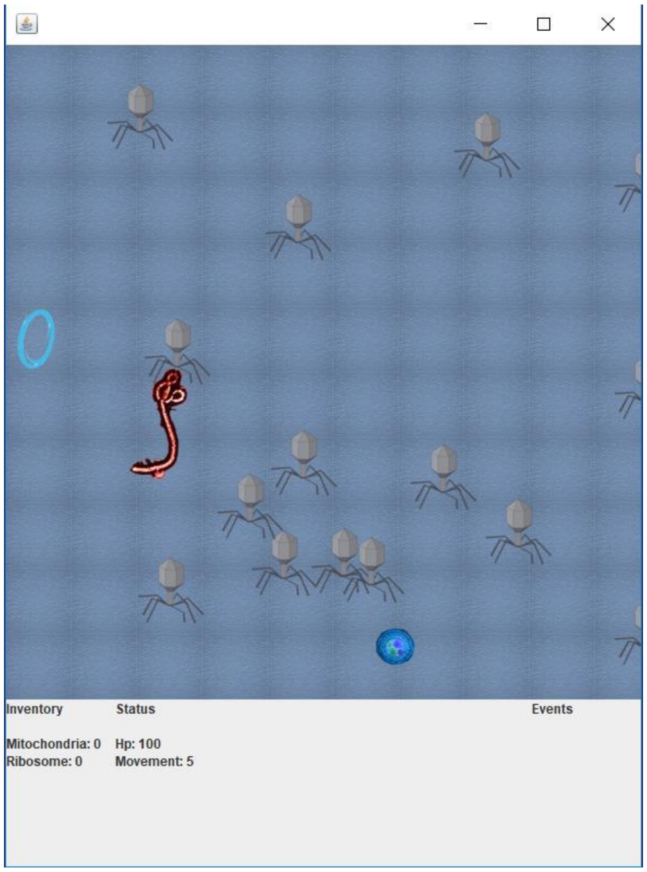
Screenshots of the Game panel:





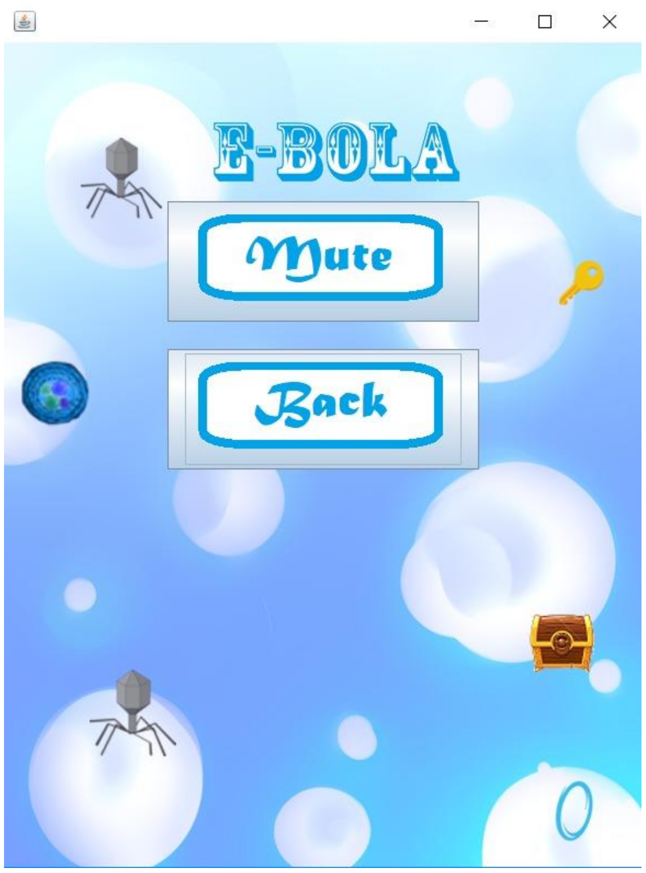





Inventory Status Events

Mitochondria: 0 Hp: 100
Ribosome: 0 Movement: 5

Screenshot of the Settings 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:

- The need of the HUD wasn't thought of before, so we had to make some changes to the **EbolaFrame** class and also add a class called **HUD**. The **HUD** shows the HP, movement speed and inventory of Celly.
- Since the state changes of interactions and **Entities** should be reported to other packages, more connections were formed between source and target packages. All these connections pass through the **GameEngine** via listeners. EntityEventListener is an example.
- Printing both images of static entities (e.g. children of **Entity** class) and dynamic entities (e.g. children of **Alive** class) to the game panel, proved to be impossible 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 wasn't 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 **EntityManager** for further actions. Functions such as **void:evaluateInput** and **boolean:checkIValidCoords** are current examples of this change.

4. Incomplete Parts

- New levels to the game could be added.
- Celly could attack the Viri rather than wait for them to expire.
- Fighting sounds could be added along with the background music. This requires threading operations and might require a more throughout design.
- Save/ Load features are yet to be implemented.

5. 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 inside "2B.E-bola"
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, navigate to 2B.E-bola file and open powershell / command line
3. Use `javac -d classes *.java; cd classes; java EbolaGame` for powershell or

   `javac -d classes *.java && cd classes && java EbolaGame` for

cmd.exe