CS 319 - Object-Oriented Software Engineering

Final Report

Galaxy Invaders

Group 3J
Umer Shamaan
Osman Can Yıldız
Denizhan Yağcı
Cemre Osan
Tanay Toksoy

Table Of Contents

1. Implementation Process

After the first iteration of the design report we started implementation immediately after creating the Github repository. We decided to work on IntelliJ as it provides a user friendly environment for coding and there is a far less chance to create a compile-time error. Furthermore, it is also easier to commit and pull code from github directly from IntelliJ. We first worked on the View Classes to make the view interfaces before adding other components. Osman was mainly responsible for those Classes. He added the appropriate gui elements. He was also responsible for creating the images and sounds of the model objects including ships, enemies and bullets. After that up until the end mid of December we started to work on the Controller Classes and Model Classes simultaneously. Umer was responsible for most of the model classes (including player, blasters, missiles, shooters, humans and boss) but others chipped in as well (for example Tanay worked on Kamikaze and Splitters, Osman worked on Bonus and Denizhan on lasers), as this was the most important and the most comprehensive part. Umer and Tany implemented most of the Controller Classes. At first the implementation process was pretty slow. We struggled to make a basic template of the game for the first demo. However, after the first demo, most of us had understood the basic structure of our code and we divided our parts more efficiently and the process was far smoother from then on. Also we had formed a basic skeleton of the code and after that we would only have to add the remaining elements to the game without having to spend much time learning about and doing research on Java swing. Furthermore, we decided that we had to make minor changes to some of the aspects we mentioned in the first

iteration of the reports due to various factors such as feedback form the ta, presence of mundane features and our own limitations. In the end, Tanay implemented the game sounds which Osman had provided us with and Denizhan worked on the shifting background.Throughout the process, we tried to stick to the OOP style as much as possible to avoid convolutions and prevent the code from becoming more complicated than it had to be. In the end, we implemented almost all of the features we had mentioned in our analysis and design reports.

2. Design Changes & Improvements

There have been a lot of changes in our game design since our first design report. At first, we were doing everything according to the design report however after some time, we realized that we had to alter the design a bit to implement certain features that we wanted to add to the game. For example we were planning to make certain classes like enemy, bonus and bullet handle the sound function by themselves however we realized that implementing sound in that way proved to be more troublesome than it was worth, so we made a class specifically to handle sound features. Just like in this example, we realized that we were depending a lot on our model manager class so we chose to use mostly singleton design pattern to continue with this project. Singleton design pattern was suitable because it made our implementation process much easier than it would have been with other design patterns even though it made finding certain bugs trickier. We also used facade design pattern, sessionListener can only access models through using modelManager as an interface. We also decided to use images with lower resolution in order to increase the performance of the game since when there were a lot of high

resolution objects on the screen, the game was barely able to run on low end computers. Finally, we added a new enemy type, splitter, which either creates 2 shooters or 1 kamikaze when it is killed.

3. Lessons Learnt

This project was perhaps the most enlightening experience in our university career up until this point. We had been taught to code but we never had to do it to such a big scale with 4 other people. At first the design and analysis reports seemed a chore but during the implementation we realized their importance. They made sure that we did not stray from what we were supposed to create. We constantly referred to them to determine the attributes and methods we would add.

We learned that there would only be a single instance of Model Manager and Game Frame class. To prevent other instances of those classes from being initialized, we used the singleton design pattern. This made sure that the instances would remain static throughout the running process of the code.

We learned how to efficiently communicate and interact with each other. In the first half of the semester, this was a big problem for us. There would be confusion among the people about their parts and our project was constantly plagued with conflicts. It is easy to code but it is very difficult to manage the development process when many people are involved. We decided that we would meet every week after Friday's lectures to decide on the implementation for the next week. We would set up agendas and divide our work.

4. User's Guide

The game will be opened by pressing jar file. For now, users have to put the code into the compiler so that they can run it. After the game is opened, the user will encounter with a menu like seen in section 4.1.

4.1. Main Menu

4.2. How To Play

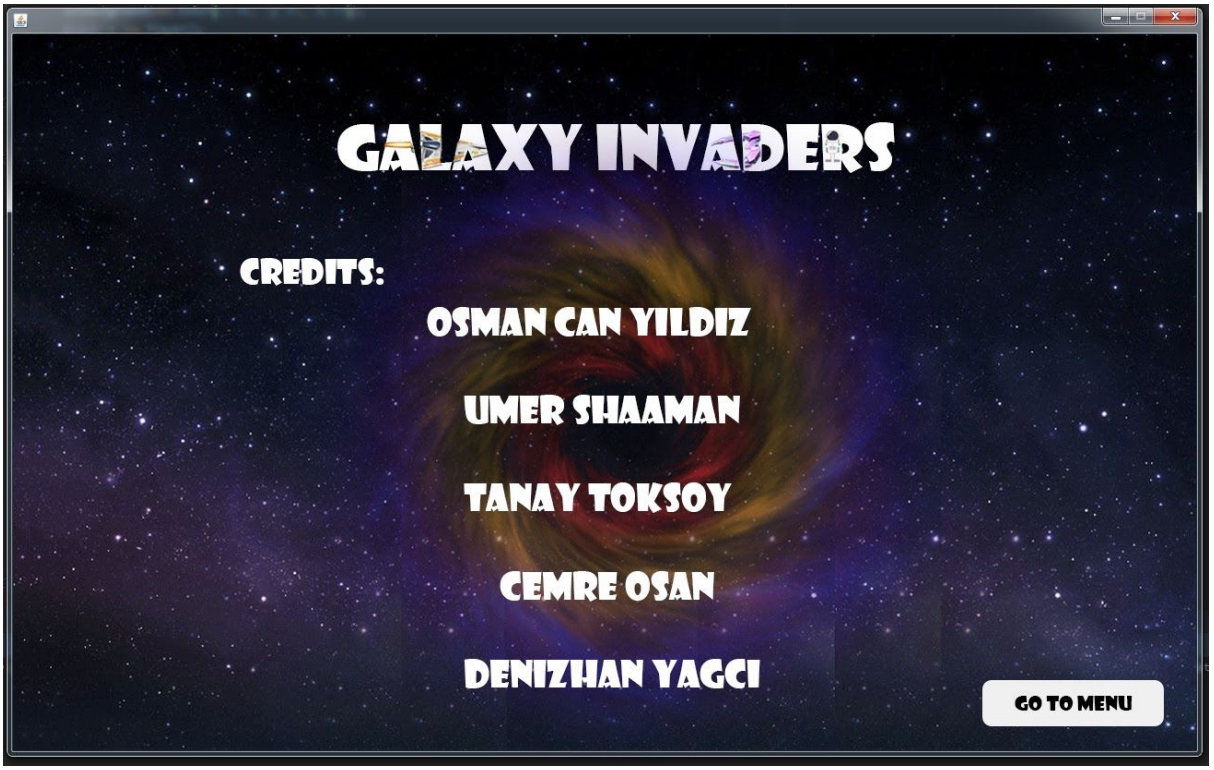
This Panel shows the basic controls of the game to the user.

4.3. Settings

Then the user goes back to main menu. In the main menu, if user clicks on “SETTINGS” button, settings about the game occurs. After changing the settings, the user can go back menu with “GO TO MENU” button like below. The change settings option is inoperational as of now.

4.4. Credits

Then the user goes back to main menu with “GO TO MENU” buttons. In the main menu, if the user clicks on “CREDITS” button, names of the creators of Galaxy invaders occur in the screen like below.



4.5. Gameplay

If the user clicks on the play button on the main menu, the game starts. The role of the player is to defeat as many enemies as possible and avoid colliding with their bullets and themselves. Some of the ships will be transporting humans and destroying those ships and catching the falling humas will grant extra score. To aid the player, different types of bonuses will be spawned after defeating some enemies. The game will culminate in a boss battle after defeating a certain amount of enemies and the final task is to defeat the boss in order to win the game.

4.5.1. Starting Screen & Labels


Health Icon


Score Icon


Human Icon


Laser Icon

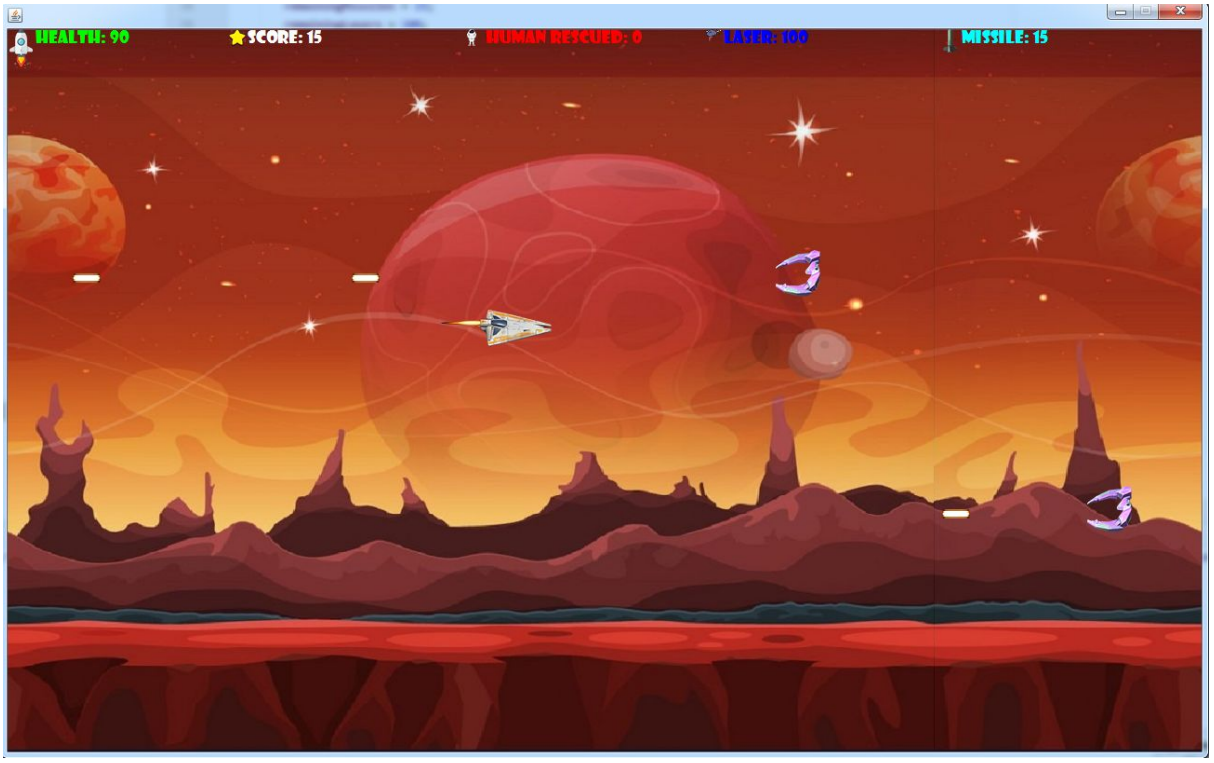

Missile Icon

When the game starts, the user interacts with the scene below. Initially, health is 100, score is 0, human rescued is 0, laser count is 100 and missile count is 15.

4.5.2. Enemy Shooter

Enemy Shooter Image

Shooter enemy type shoots orange laser which deals 10 damage to player's spaceship. Shooter's collision with spaceship deals 10 damage to the spaceship. Shooter has also 10 health.



4.5.3. Enemy Shooter With Human

Enemy Shooter With Human Image

These types of shooters will be carrying humans. After destroying them the humans will be released. The will have to be caught before they hit the ground to get +100 scores.

4.5.4. Enemy Kamikaze

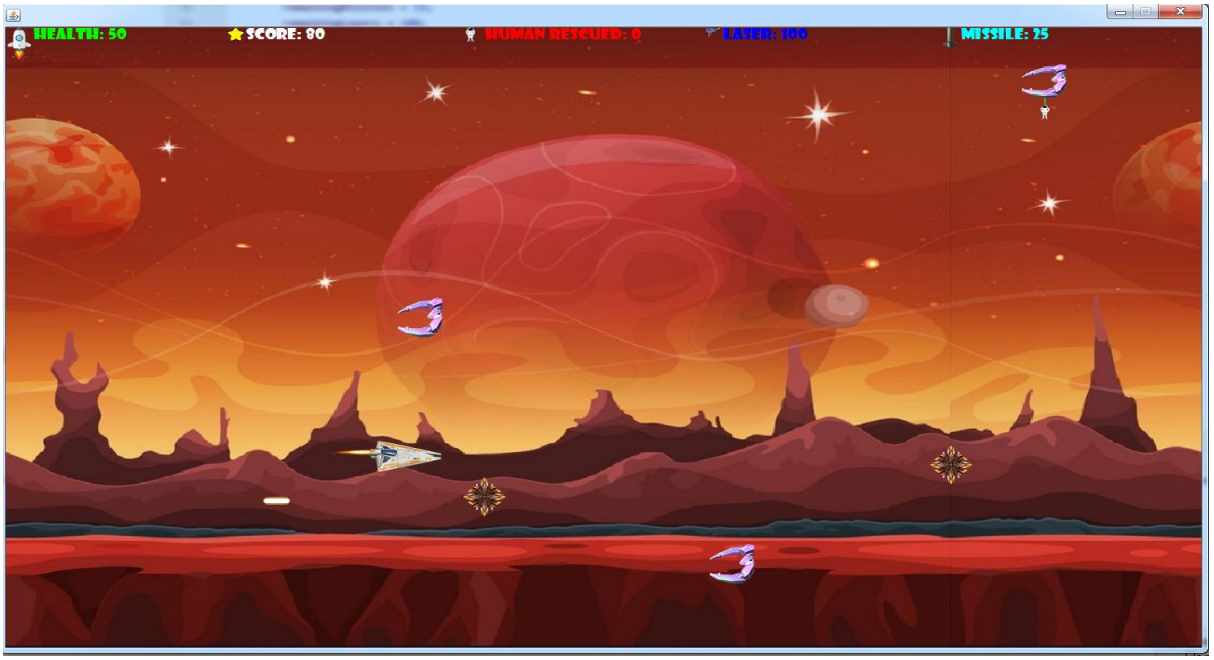
Enemy Kamikaze Image

Kamikaze does not shoot but they move aerobically with the purpose of colliding with the player's spaceship. The collision with spaceship deals 30 damage to the player's spaceship. Kamikazes are really fast. Their health is 10.

4.5.5. Enemy Splitter

Enemy Splitter Image

Splitter enemy type shoots orange laser which deals 10 damage to player's spaceship, the same as shooter type of enemy. Splitter's collision with spaceship deals 10 damage to the spaceship. Splitter has also10 health. Splitter's special skill is that when Splitter is destroyed, 2 shooters or a kamikaze are spawned at the position.



4.5.6. Firing Blaster

Blaster Image

This is a basic player bullet which does 10 damage. The player will have an infinite amount of these. However their speed is less when compared to other bullets.

4.5.7. Firing Laser

Player Laser Image

These lasers are similar to the bullets but the difference is that they are not destroyed when they hit a target. They also have higher speed.

4.5.8. Firing Missile

Missile Image

The missile will create a lock on the nearest enemy it finds. It also has 20 damage and high speed. However, the amount of the lasers is limited at the start.

4.5.9. Bullet Bonus

Bullet Bonus Image

This bonus will either grant the player either 10 additional missiles or 50 additional laser bullets.

4.5.10. Health Bonus

Health Bonus Image

This bonus will grant the player with a health upgrade of 100 points.

4.5.11. Speed Bonus

Speed Bonus

This bonus will increase the speed of the player by 2 folds given that the player ship has not reached a maximum shield threshold.

4.5.12. Shield Bonus

Shield Bonus

This bonus will provide a layer with a protective layering for a short amount of time which will prevent the player from receiving any amount of damage from either bullets or collision with other ships.

4.5.13. Boss Level

Boss Image

The boss is spawned after defeating a certain amount of enemy ships. The boss is more difficult to defeat than other enemies as it has a higher health and fires about three bullets at the same time.

5. Build Instructions

1. Clone and download the Github code from the repository named CS319-3J-DE to the desired folder.
2. Download and Install IntelliJ if not already done so.
3. Download and install java 8 if not already downloaded.
4. Start IntelliJ and click in ‘Open Project’.
5. Navigate to the project folder and double click on the project file.
6. Click on ‘run’ icon on the nav bar.

5.1. System Requirements

- Processor: Core i3 or above
- Ram: 1Gb or above
- Operating System: Windows 7 or above
- KeyBoard and mouse

6. Individual Parts

Umer Shamaan:

- Analysis Report:
    - Non-Functional Requirements
    - Use Case Models
    - Sequence Diagrams (partial)
    - Class Diagram (partial)
- Design Report:
    - Trade-offs
    - Class Diagram (partial)
    - Design Decision Patterns (partial)
    - Packages
- İmplementation:
    - Player
    - Enemy, Enemy Shooters, Boss
    - Bullet, Bullet Missile, Bullet Blaster
    - Human
    - Controller Classes(Partial)
    - View Classes (Partial)

Tanay Toksoy:

- Analysis Report:
    - Activity Diagram
    - Sequence Diagrams (partial)
    - Class Diagram (partial)
- Design Report:
    - High Level Software Architecture
    - Class Diagram (partial)
- Implementation:
    - Kamikaze, Splitter
    - Sound
    - SessionListener (partial)
    - GameSceneView (partial)

Osman Can Yildiz:

- Analysis Report:
    - Overview
    - Functional Requirements
    - Class Diagram (partial)
- Design Report:
    - Purpose Of System & Design Goals
    - Criteria
    - Class Diagram (partial)
- Implementation:
    - Bonus
    - Creating Image and Sound Assets
    - View Classes (partial)
    - Controller Classes (partial)

Denizhan Yağcı:

- Design report class diagram definitions
- Introduction to design report
- Laser Bullets
- Transitioning Background
- Trailer Videos

Cemre Osan:

- Analysis Report 1:
    - Activity Diagram
- Analysis Report 2:

    - Use Case (partial)
    - Activity Diagram (partial)
    - Non-functional Requirements
- Design Report 1:
    - Introduction and Design Goals