Bilkent University

Department of Computer Engineering

CS319 Term Project

Defender: Fiyuv++

Analysis Report Iteration 2

Doğukan KÖSE, Hamza PEHLİVAN, Musab OKŞAŞ, Meryem EFE, Aybüke ERTEKİN

Instructor: Eray TÜZÜN

Teaching Assistants: Barış ARDIÇ, Alperen ÇETİN, Hasan BALCI

Contents

Analysis Report

Defender: Fiyuv ++

1. Introduction

Defender is an old arcade game whose first version was released by Williams Electronics in 1981 [3]. As a new generation, it was very funny to experience a video game developed before our birth. Even if the game was developed approximately fifty years ago, we were impressed by its creators. We thought that it could be very fantastic to add new features to this game, implement them using Object-Oriented design, and get people try our new version of Defender. Therefore, we chose Defender as our term project game we named our new version of the game Fiyuv++.

Our version of Defender will have following features:

- Single Player Mode
- 2 Players Mode
- Sound Options
- Spacecraft Options
- Background Color Theme Options
- Different Backgrounds in Different Levels

- Levels with Increasing Difficulty
- Various Enemies with Different Characteristics
- Buffs
    1. Increasing Speed
    2. Smart Bombs
    3. Increasing Power of Attack
    4. Increasing Fire Rate
    5. Multiple Shot
    6. Hyper Jump
    7. Shield
    8. Health Point
- A Radar Displaying the Place of Enemies
- Bosses at the end of Levels

A player will be able to control a spacecraft in order to move, attack enemies which are spreading from enemy stations, and eliminate their space station. The main purpose of the game is to destroy enemy stations, and defeat the boss of each level to protect the world. In the beginning, the player will be able to play first level of the

game and other levels will be locked. If the player passes a level, a harder level will be unlocked. After destroying all enemy stations, the player has to defeat boss of the level which shows up at the end of each level. Namely, each level will have two subsections. In the first section, the player confronts enemies and tries to destroy enemy stations. If the player is able to complete first section by destroying all enemies and enemy stations, second section will start and the player will confront boss of the level. Bosses will have different qualities in each level. We call first section of the game “PreBossScene” and we call second section of the game “BossScene”. We will use these words throughout the report.

Our game is keyboard controlled. The players will have ability to change key bindings of the game from settings. The game will also have mode for two players. In this mode, two players will fight against enemies together.

JavaFX will be used to implement the game. We will try to design and implement our game using object-oriented programming principles.

2. Overview

2.1. Gameplay

Fiyuv++ is a battle game which has a story passing in the space. The player controls his/her own spacecraft and fights against enemy forces. To win the battle and protect the world, the player has to finish different levels successfully. Each level consists of enemy spaceships, enemy space stations where enemy spaceships can spawn, and boss of the level. Aim of the player is to defeat the boss of the level after s/he gets rid of all enemy ships and enemy stations. When the player demolishes enemy forces, s/he gains points. The player can take advantage of different buffs which are obtained from destruction of enemy spaceships. After all enemy forces are destroyed, the player encounters the level's boss to finish the level.

2.2. Map

The area where the game is played is called map. In Fiyuv++, there are 2 different maps: pre boss and boss map.

In the first scene which the player has not faced with the boss yet, the game is played in the pre boss map. The map includes all game entities such as players' spacecrafts, enemy spaceships, enemy stations, and meteor showers.

In the second scene which the player face with the boss, the game is played in the boss map. There will be players' spacecrafts and meteor showers as well along with the bosses of each level in the boss map.

2.3. Initial Properties of a Player's Spacecraft

The spacecraft has following properties at the beginning of the game:

- 100% health point which can decrease due to attack of enemy spaceships, bosses, and hit of meteors.

- A gun which has definite power, period, and single shot ability. Initially, the gun power is 10 unit and the gun period is 1 second.

- Constant velocity when spacecraft moves. The spacecraft cannot move by accelerating. Initial velocity is 20 unit.

- Rechargeable hyper-jump ability which teleports the spaceship to some distance through its direction. This distance depends on charge level of hyper-jump ability. If its battery is 100% full, the spacecraft can go forward 100 units. If its battery is 50%, the spacecraft goes forward 50 units. As a result, the spacecraft can teleport to a distance proportional to hyper-jump battery. However, if its battery is less than 25%, the player cannot use hyper-jump ability. In the beginning, the player have 100% hyper jump charge.

- 3 smart bombs. Smart bombs decrease health point of every enemy force on the visible map by 100. The player can have at most 3 smart bombs.
- A radar which provides the player to detect enemy forces.

2.4. Modes

There are single player and 2 players modes. There is no change in 2 players mode except that the players fight together against enemies.

2.5. Buffs

In the game, there are 8 different buffs which can make the spacecraft get a new or improved feature / ability. These buffs can be gained after demolishing enemy spaceships.

Buff Types:

- Hyper-jump Charge
- Extra Smart Bomb
- Health Buff
- Shield Ability

- Speed Buff
- Multiple Shot Ability
- Increased Attack Speed
- Increased Attack Power

Features of buffs are explained in detail in section 3.2.

2.6. Enemy Forces

2.6.1. Enemy Spaceships

There are 3 tiers of enemy spaceships. There are simple and evolved versions of each tier of enemy spaceships.

Evolved and simple versions of a certain type behave in the same way. The only difference is evolved versions of an enemy tier are stronger than simple versions as its name suggests. This strength is because of that most of unevolved enemies' features are doubled in evolved versions.

Enemy tiers will be explained in detail in the section 3.7.1.

2.6.2. Enemy Stations

Enemy Stations are the places where enemy spaceships spawn at a certain frequency. There are two categories of enemy stations. In first category, Simple Versions of Tier 1, Tier 2 and Tier 3 enemy ships are created. In second category, Evolved Versions of Tier 1, Tier 2 and Tier 3 enemy ships are created.

Simple Version enemies can go to the stations where evolved versions are created in order to make themselves stronger and “evolved”.

2.6.3. Bosses of the Levels

After the player destroys all the enemy spaceships and stations, boss of that specific level shows up. The boss is the last challenge for players to complete level. Each boss has unique abilities different from each other. These unique abilities will be explained in the section 3.7.3.

2.7. Settings

In this menu, the player can adjust sound volume, key bindings related to spacecrafts. In this way, for spacecrafts’ actions, customized keys can be assigned. Lastly, to be more user-friendly, there are color themes for color-blind people and for normal people as well.

2.8. Extended Features

Before continuing with functional requirements, we wanted to explain what we added to Defender game in our Fiyuv++.

1. **3 Levels:** In Defender game, there is not any level. There is just continuous flow of one game. We thought that it could be beneficial to add levels and make players feel satisfied for reaching the end and the success.
2. **Bosses at the end of each level:** We thought that these bosses will make the game more challenging.
3. **Enemy Stations:** This stations add a new goal to the game. It is not enough to destroy enemies. The player has to destroy enemy stations to win the game.
4. **Meteors:** Meteors are very compatible with the concept of space. We thought that they increase reality of the game.
5. **Buffs:** These buffs make the game more playable. We already increased the challenge of the game. We thought that little helps to the player would be beneficial.
6. **Health Point:** Spacecraft, enemy stations, enemies have health points. It is more challenging to have enemies that are not destroyed by only one bullet.
7. **Background Color Theme Options:** These color options would be beneficial to people who are not comfortable with the color that we choose.
8. **Spacecraft Appearance Options:** Players will be able to choose which spacecraft they want to use.

Note: We removed humans in the space which original version of the game contains. We thought that it will be difficult for players to keep up with the game if we do not remove humans from the game. Because we already have various new functionalities such as enemy stations, buffs or bosses, it would be beneficial to remove humans in order to increase simplicity of the game.

3. Functional Requirements

3.1. Play Game

The game will have one player and two players modes. The player has to choose one of these options from main menu to start the game.

3.1.1. Single Player Game

After choosing this option, the game will not immediately start. Firstly, the player has to choose the outlook of the spacecraft and the level s/he wants to play. In the beginning, all levels will be locked except the first level. After passing a level, the next level will unlock. However, the player can always play a previous level by choosing it in this stage. After clicking “Play”, the game will start.

In this mode, the player has to fight against all enemies alone. The spacecraft can move only in right and left direction. However, the player has ability to arrange altitude. Spacecraft can fire in the direction of movement. The player tries to destroy all enemy stations using these abilities. The player will be able to find the places of enemy stations by looking on the radar. While doing this, there will be enemies spreading from their space stations and trying to eliminate the player’s spacecraft. The player has to shoot these enemies and escape from getting hit by them. If the player gets hit, spacecraft loses some of its health points. After destroying all enemy stations, the player has to destroy the boss of this level. If spacecraft loses its all health points, the game ends unsuccessfully. To increase success possibility in the

game, the player can use buffs that show up after eliminating enemies or hitting bosses. At the beginning of the game, the player has %100 charge for hyper jump and 3 smart bombs. The qualities of these abilities and the buffs are explained in detail in section 3.4. The player uses arrow keys for movement, “Z” for hyper jump and “X” for smart bomb as default. If the player is not comfortable with these keys, s/he can arrange them from settings.

3.1.2. Two Player Game

If the player chooses this option from the main menu, the game will start with two spacecrafts. Players have to choose appearances of their spacecrafts before starting the game.

In two-player mode, the map will be horizontally divided into two. Player 1 can move in the upper part of the map and Player 2 can move in the lower part of the map.

Two players will fight against enemies together. Their health points will be independent from each other. If one of the spacecraft loses all health points, the other one will be able to continue to the game. Also, buffs they have will be independent from each other and they use and take their buffs separately. Each spacecraft will have all qualities that a spacecraft has in one player mode such as 100% charge of hyper jump, 100 health points, and 3 smart bombs. However, their scores will be calculated together.

In two-player mode, player one uses arrow keys for movement, “Z” for hyper jump and “X” for smart bomb as default. Player two uses “W”, ”A”, “S”, “D” for up, left,

down and right movement, s/he uses “1” and “2” for hyper jump and smart bomb as default. If players do not feel comfortable with keys, they can change these keys from settings.

3.2. Levels

In our game, we will have 3 levels with increasing difficulty. This increase is because of new type of enemy introduced in each level, increase in the the spawning frequency of enemy stations and different qualities of bosses of each level. In level 1, enemy stations will only spawn tier 1 enemies. In level 2, they will spawn both tier 1 and tier 2 enemies randomly. In level 3, enemy stations will be able to spawn tier 1, tier 2 and tier 3 enemies randomly. Spawning frequency of space stations will be 1 enemy / 5 seconds in level 1, 1 enemy / 4 seconds in level 2, and 1 enemy / 3 seconds in level 3. Enemy stations would be two different types: simple enemy station or evolved enemy station. Simple enemy stations will spawn simple enemies of specified tiers, while evolved enemy stations spawn evolved enemies. However, they will have the same spawning frequency. Also, bosses that a player confronts at the end of each level are different from each other. Their qualities are explained in section 3.7.3.

Level 2 and 3 will be locked at first. If the player succeeds to pass level 1, s/he will be able to play level 2 and if s/he passes level 2, s/he will be able to play level 3.

Each level will have a different background.

3.3. Map

The Map is where the game is played. Namely, it is where spacecraft moves, fires, and all other entities are found. Map in the “PreBossScene” and “BossScene” will have different qualities.

3.3.1. PreBossScene Map

The game is played in the area whose width is 10000 units and the height is 1000 units. This whole area is called as “PreBossScene Map”. However, the player can see a small part of the map in the screen size. Even though the player can move in the map in order to see the other parts of the map, s/he can also see places of the enemy forces on the radar which is placed at the top of the screen.

The map includes all game entities. First of all, there are player’s spacecraft and enemy tiers. There are also enemy stations in various places of the map which will be determined randomly before the game starts. Lastly, meteor showers will take place in the map.

3.3.2. BossScene Map

The size of the map in “BossScene” will be limited with the screen size. This map does not contain any enemy or enemy station. On the screen, there will be the

spacecraft, the boss of the level and meteors. Also, the radar will not take part in this map. In this map, spacecraft can move inside a ¾ screen width distance and the boss will move inside remaining ¼ screen width distance. Namely, the spacecraft and bosses can not touch each other in this map.

3.4. Buffs

In the game, we will have 8 types of buffs. These buffs will show up when an enemy is eliminated. When an enemy is created, which buff will appear after it is destroyed will be randomly determined, but will be hidden. Some enemies also may not have any buff. The probability that a buff will show up after an enemy is destroyed is 3%. Also, buffs may appear when players are in the BossScene and they hit the boss. However, the probability that a buff will appear when the player hits boss is dependent on the level. In level 1, the probability is %3, in level 2, it is %2 and it is %1 for level 3. Also, in the BossScene, the player can not use or take hyper jump buff, because the player will have to stay in the map that has the same size with the screen size. The player will be able to take these buffs by just passing over them. Buffs will disappear from the map after 10 seconds if the player does not take the buff. According to the qualities of buffs, we can separate them into three categories.

In the first category, there is only shield buff. This buff has a time limit. When the player takes this buff, the buff will be used immediately and it will last 5 seconds. If

the player takes another shield buff before time is not up for a shield buff in use, the time for the buff will be reset into 5 seconds.

- **Shield:** While using this buff, spacecraft can crash into everything except enemy stations without losing any health point. While using this buff, if spacecraft crashes into a meteor or an enemy, what spacecraft crashes will be eliminated. However, if spacecraft crashes into an enemy station, spacecraft will be eliminated even if it uses shield buff. If it takes another shield buff before time is up, time for shield buff will be reset. If the player uses this buff in BossScene, the player will be protected from the bullets of the bosses or from meteors. However, the player can not escape from special abilities of bosses using this buff.

In the second category, buffs improve qualities of spacecraft. These improvements are irreversible. If spacecraft takes one of these buffs, it will continue with these improvements until the end of the level.

- **Multiple Shot:** At the beginning of the level, when spacecraft fires, it shoots one bullet at once. If spacecraft takes this buff first time, it will shoot two parallel bullets at once. If spacecraft takes multiple shot bonus second time, it will shoot three parallel bullets at once. After this, spacecraft can not start to shoot more than three bullets, even if it takes another multiple shot buff.

- **Increased Attack Power:** At the beginning, spacecrafts' bullet gives 10 health points damage when it crashes into a meteor an enemy or an enemy station. If the player takes an increased attack power buff, this damage will be

20 health points and if the player takes one more increasing attack power buff, it will be 30 health points. The maximum attack power is 30 health points.

- **Increased Attack Speed:** At the beginning of the game, spacecraft fires 1 bullet for each ½ second if the player continuously presses fire key. Namely, gun period is ½ second. If spacecraft takes an increased attack speed buff, its gun period will be ⅓ second and if it takes one more increased attack speed buff, its gun period will be ¼ second. The minimum gun period is ¼ second.

- **Speed:** This buff increases the speed of the spacecraft. The default speed of spacecraft is 20 units/sec and it will be doubled if spacecraft takes speed buff. If spacecraft takes another speed buff, its speed will increase to 60 units/sec and it will continue with this speed until the end of the level.

In the third category, there are buffs to update hyper jump, smart bomb abilities of the spacecraft and spacecraft's health point. At the beginning of the game, the player has 100 health points, 3 smart bombs and 100% charge for hyper jump. However, as time is spent, the player may use smart bomb or hyper jump and may lose some health points. These buffs are useful to refresh them.

- **Smart Bomb:** At the beginning of the game, the player has 3 smart bombs. These buffs will be displayed in a slot on the upper side of the map. If the player uses a smart bomb, everything at a distance of 1000 units from spacecraft loses 100 health points except spacecraft and a smart bomb will be reduced from the slot. If the number of smart bombs spacecraft is less than

3, spacecraft gains one more smart bomb if it takes this buff. If it is 3, spacecraft can not take another smart bomb.

- **Hyper jump:** The player has 100% charge for hyper jump at the beginning of the game. If the player uses hyper jump, it will move 100 units immediately. This will make charge of hyper jump zero. After this, hyper jump will be recharged as time is spent. Hyper jump will be recharged %1 for every second. If the charge of hyper jump becomes more than %25, the player can use this ability, but it will move only 25 units. Hyper jump buff fully charges hyper jump ability. This buff can not show up in BossScene, because in BossScene, the player cannot use hyper jump ability.
- **Health Point:** The player's health point is 100 at the beginning of the game. However, it can be decreased by enemy bullets, meteors or enemies which crash into the spacecraft. This buff adds 15 health points to the health point of the spacecraft. The health point can be at most 100.

3.5. Meteors

Meteors will have circular shapes. They appear in the map randomly and they move downward from the upper side of the map. They are eliminated if they crash into an enemy, spacecraft or an enemy station. When they crash into these objects, they lose health points. They can also be eliminated by the bullets of spacecraft or enemies. They can have various radii from 10 units to 100 units. Their damage which they give to the objects when they crash into them and their speed will be scaled according to the size of meteors. If the meteor's size is large, it will cause

more damage and its speed will be lower. If the meteor's size is small, it will cause less damage and its speed will be higher. For example, if a meteor which has 10 units radius gives 3 health points damage and its speed is 50 units/second, a meteor which has 100 units radius will give 30 health points damage and its speed will be 5 units/second.

3.6. Bullets

In the game, spacecraft, enemy tier 1, enemy tier 3, evolved enemy tier 1, evolved enemy tier 3, boss 1, boss 2 and boss 3 have firing ability. The speed of bullets is the same for all objects and it is 10 units/second. However, the period and the damage they give are different for each entity and they are explained in the subsection of each entity. A bullet will be eliminated from the map, after going 1000 units distance.

3.7. Enemy Forces

3.7.1. Enemy Spaceships

There are 3 tiers of enemy spaceships. There are simple and evolved versions of each tier of enemy spaceships.

Note that evolved and simple versions of a certain type behave in the same way. The only difference is evolved versions of an enemy tier are stronger than simple

versions as its name suggests. This strength is because of that its features except the velocity are doubled.

Tier 1: Enemies in this tier are basics. They are easy to kill compared to other types. They do not damage the player's spacecraft until they see it with their limited vision. If an enemy tier 1 doesn't see the player's spacecraft, it wanders randomly.

Enemy tier 1 has 10 health points. When it hits spacecraft, spacecraft loses 2 health points. Namely, its gun power is 2 health points and the gun period is 1 bullet/second. The velocity of tier 1 is 30 units/second. The radius of vision is 10 units.

Evolved Tier 1: When an enemy tier 1 is evolved, its behaviour doesn't change. It still wanders randomly. If it sees the player, it stops and fires to the player's spacecraft. However, there are changes in features. Its health point becomes 20. Its gun power is 4 health points and gun period is 2 bullets/second. Its vision has 20 units radius.

Tier 2: Enemies in this tier are like suicide plans. Normally, they wander with 20 units/second velocity. However, when they see the player, they use their high speed (50 units/second) to crash into the player's spacecraft. However, they move in the direction which they see spacecraft first time. They can see the spacecraft from 30 units distance. If spacecraft goes more than 30 units distance from these enemies, they leave to trace spacecraft and return to their 20 units/second velocity. Also, there is a time limit to trace spacecraft. If these enemies trace spacecraft for 10 seconds and they still could not crash into spacecraft, they leave tracing and continue with 20

units/second velocity. They have the highest damage capability among all types. If they are able to crash into spacecraft, the damage which they give to the spacecraft is 50 health points.

Evolved Tier 2: The Evolved version of tier 2 behaves in the same way. However, it has 40 health points. Also, the damage they give is 100 health points, so evolved tier 2 can destroy the spacecraft if it can crash into the player's spacecraft. It can see the spacecraft from 60 units distance.

Tier 3: These enemies fight against the player from 50 units distance by using their shooting ability. They are very hard to kill and the most challenging enemy spacecrafts. Once they are destroyed, they split into Tier 1 and Tier 2 enemies according to their versions. It means if a Tier 3 – Simple Version enemy is destroyed; it splits into Tier 1 – Simple Version and Tier 2 - Simple version enemies. Similarly, if a Tier 3 – Evolved Version enemy is destroyed, it splits into Tier 1 – Evolved Version and Tier 2-Evolved version enemies. An enemy in this tier has 50 health points. Its gun power is 10 health points. Its velocity is 15 units/second. Also, its vision has 50 units radius.

Evolved Tier 3: The Evolved version of tier 3 is the most powerful enemy except bosses. It has 100 health point. Its gun can give 20 health points damage. Evolved tier 3 can see the player from 100 units distance.

3.7.2. Enemy Stations

Enemy stations are the places where enemies spawn at a certain frequency. There are two categories of enemy stations.

Simple Enemy Stations: A simple enemy station has 500 health points and it creates one enemy from each simple versions (Simple Tier1, Tier2, Tier3) in every 5 seconds. Simple enemy stations spawn different kind of enemies according to level. For example, in level 1, it only creates tier 1 enemies. In level 2, it creates tier 1 and tier 2 enemies and in level 3, it creates tier 1, 2 and 3 enemies. In the map, there are 20 simple enemy stations at the beginning of each level.

Evolved Enemy Stations: An evolved enemy station has 1000 health points and it creates one enemy from each evolved versions (Evolved Tier1, Tier2, Tier3) in every 3 seconds. Evolved enemy stations spawn different kind of enemies according to level. For example, in level 1, it only creates evolved tier 1 enemies. In level 2, it creates evolved tier 1 and evolved tier 2 enemies and in level 3, it creates evolved tier 1, 2 and 3 enemies. Also, if an simple enemy stop off in the evolved enemy stations, it evolves and becomes stronger enemy. In the map, there are 10 evolved enemy stations at the beginning of each level.

3.7.3. Boss of the Levels

After the player destroys all the enemies and enemy stations, the boss of that specific level shows up. It is the last challenge for players to complete the level. Each boss has unique abilities different from each other.

Boss 1: Boss 1 is the boss of the first level. It has 1000 health points. Also, it has a gun whose power is 10 units and the period is 2 bullets/second. The gun's bullets go from left to right in the linear direction. The unique ability of Boss 1 is the laser. It throws laser with 1% probability. If the player's spacecraft touches the laser, it is immediately destroyed and the player loses.

Boss 2: Boss 2 is the boss of the second level. It has 1500 health points. Also, it has a gun whose power is 20 units and the period is 3 bullets/second. The gun's bullets go to the spacecraft in the linear direction. The unique ability of Boss 2 is the rocket. It throws 5 rockets to 5 different areas which are determined randomly with 2% probability. Each of these areas has 50 units radius. And the damage which each rocket gives is 50 health point.

Boss 3: Boss 3 is the boss of the third level. It has 2000 health points. Also, it has a gun whose power is 30 units and the period is 1 bullets/second. The gun's bullets go to the spacecraft in the linear direction. The unique ability of Boss 3 is the little boss. It throws little bosses with 3% probability. Little Boss is like a different type of bullet. These little bosses rebound from the limits of the screen. Little bosses disappear when they hit the wall 3 times. The damage which a little boss gives is 50 health points.

3.8. Score Calculation

When the player starts the level, s/he will have 0 points. The player gains 2 points for every destroyed tier 1 enemy, 4 points for every destroyed tier 2 enemy and 6 points for every destroyed tier 3 enemy. If spacecraft eliminates evolved versions of these enemies, the player gains twice as many points as simple version of this tier. If spacecraft eliminates a simple enemy station, it brings 50 points to the player and if it eliminates an evolved enemy station, it brings 100 points. If spacecraft confronts the boss of the level, the player gains 10 points for each time spacecraft hits the boss. If the boss is destroyed, the player gains 1000 points. The player loses 5 points for every 10 seconds spent while playing the game in PreBossScene. Namely, the player has to complete PreBossScene as quickly as possible to get high score. However, the score cannot be a negative number. Therefore, if the player continues to lose points even if s/he has 0 point, the score will stay the same. The score will be calculated separately for each level.

3.9. Menu - How to Play

The player can access this screen from the main menu. This screen contains information about

- Rules
- Different Enemy Types

- Bosses of Each Level
- Keyboard Controls
- Buffs

3.10. Menu - Options

The player can access “Settings” from Main Menu. This screen contains options for:

- Adjusting the volume
- Key settings
- Background color themes

The player also has options for the appearance of the spaceship and the level explained in 3.1.

3.11. Menu - Credits

This is one of the options in main menu. If the player chooses this option, s/he will be able to see information about people who contributes to the development of the game.

3.12. Menu - High Scores

The player can access ten of the highest points people reached. Each level will have its own list. Therefore, this section will contain 3 highest score lists. This quality will make people more competitive to get higher scores and exceed previous scores.

3.13. Pause

After the game starts, the player can pause the game by pressing “ESC” key. When the player pauses the game, a menu will show up. The player will be able to continue to the game, change settings, and exit the game.

3.14. Additional Requirements

After iteration 1, we have not added any new functionality to the game.

4. Non-functional Requirements

4.1. Game Performance

Our game will have a good quality of performance since we are planning to use JavaFx which has “high performance graphics engine, called Prism” [1]. We expect our game to have **45+ fps** in the system described above:

- CPU: Intel Core i5-2500K / AMD FX-6300
- RAM: 8 GB
- OS: Windows 7 SP1
- VIDEO CARD: Nvidia GeForce GTX 770 2GB / AMD Radeon R9 280
- PIXEL SHADER: 5.0
- VERTEX SHADER: 5.0
- FREE DISK SPACE: 150 GB
- DEDICATED VIDEO RAM: 2048 MB

Especially, with the systems with better features, the game will have better performance. We are planning to design the animations to be as smooth as possible for lower systems also.

4.2. Portability

Our game will be suitable for different operating systems and different hardware due to one of the most beneficial features of Java which is working on every environment with proper Java virtual machine. “A Java virtual machine (JVM) is a virtual machine that enables a computer to run Java programs as well as programs written in other

languages that are also compiled to Java bytecode" [2]. Our game will be tested in **the 3 operating systems** listed above:

- Windows 10
- Mac OS X 10.11
- Ubuntu 15.10 (Wily Werewolf)

4.3. Extendibility

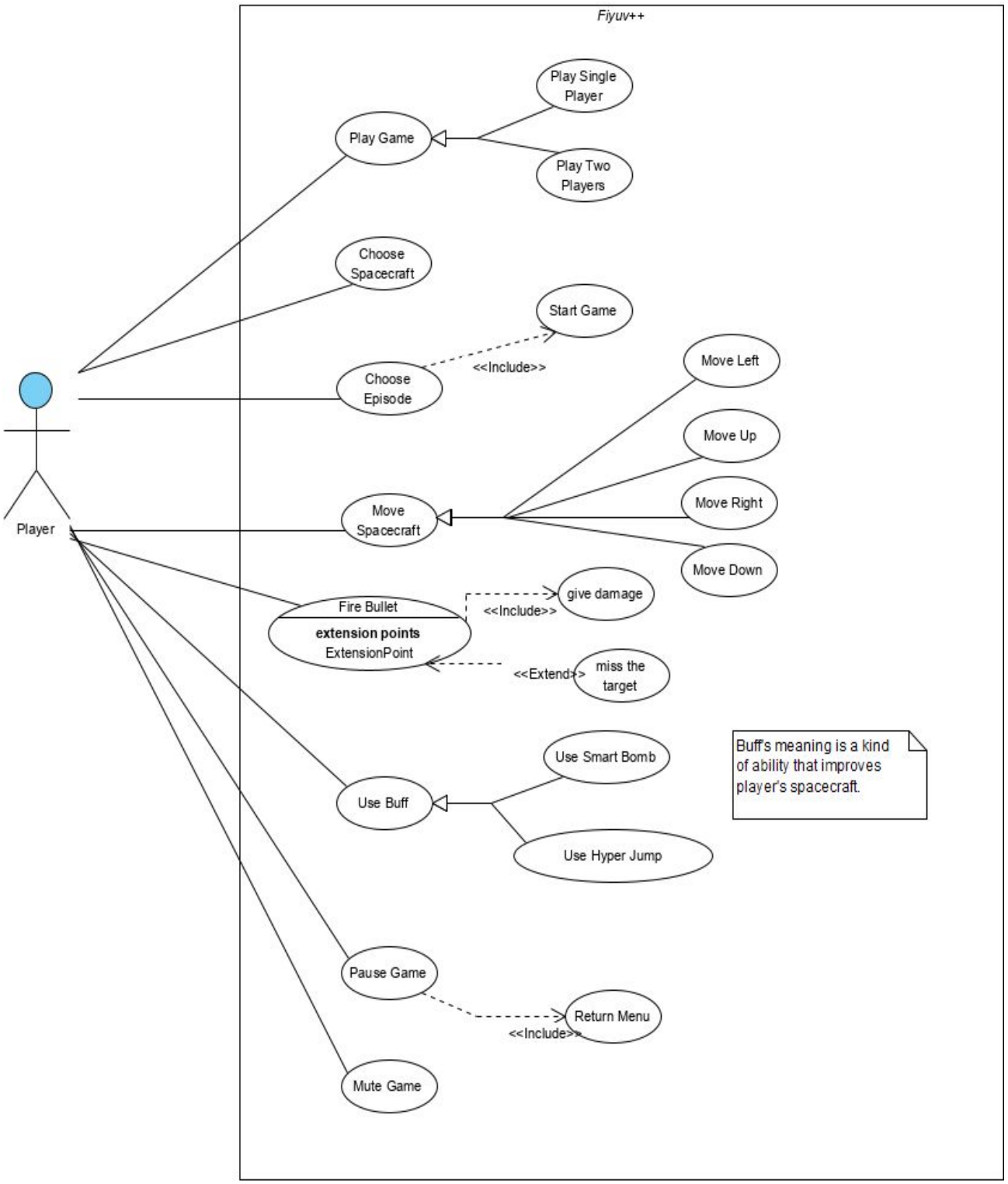
Our game is designed suitable to add new features or extend existing features easily without having major bugs or obstacles for next versions. When there will be new feature to be added into our game, **at most 3 classes** will be affected by this. In order to achieve this goal, we designed our object class diagrams compatible for prospect new codes as it can be seen in section 5.4.1. For example, if there were a new enemy; by inheriting from Enemy class, this new enemy would already have core features such as health, radar, damage, destination, or even location and hitbox from LocatableObject class without writing extra codes. Other than our game design, our game can be extended to other platforms such as web or mobile. As we are using JavaFx which works on these platforms as well, we would not have to design our project from beginning.

4.4. Usability

Our game will be usable by people who are **13+ years old with no disabilities** except color-blindness and who are **proficient in reading english**. For color-blind people there will be **at least 3 color** themes.

5. System Models

5.1. Use Case Model of the Game



General Description

When the game is started, player faces with the game menu which provides options for the player. The Player can **choose single player or 2 players options to play game**. According chosen option, the game provides different choices for user to **choose his/her spacecraft**. After the spaceship is selected, the game displays different level options for user, so that user can **choose a level**. However, player cannot select higher levels without passing lower levels. After spacecraft and level is determined, the player can start to play game. In the game user can **move his/her spacecraft** to **up, down, right, and left** directions. Also, **the player can fire** with his/her spacecraft against enemies' spaceships. The player also can **use buffs** to improve the spacecrafts features. While the player is in the game screen, s/he can **pause the game** or **mute the game**.

Use Case #1

Use Case: Play Game

Participating Actor: Player

Pre-conditions:

· Player must be in the main menu to select single player or 2 players options

Post-conditions:

· Player is successfully directed to spacecraft selection part.

Main Event Flow:

· Player who is in the main menu clicks the Singleplayer or Two Players buttons

· System responses and displays spacecraft selection menu for user

Alternative Event Flow(**s**):

· Other Menu Options

Use Case #2

Use Case: Choose Spacecraft

Participating Actor: Player

Pre-conditions:

· Player should have decided that s/he plays the game whether two players or single player

Post-conditions:

· spacecraft that will be shown in the game is successfully determined

Main Event Flow:

· Player chooses a spacecraft among given options

· System stores information of the chosen spacecraft to display it in the game

· System displays the level selection menu to player/s

Alternative Event Flow(**s**):

· Player can back to menu without choosing spacecraft

Use Case #3

Use Case: Choose Episode

Participating Actor: Player

Pre-conditions:

· Player must have chosen spacecraft to choose level

Post-conditions:

· System successfully starts the game

Main Event Flow:

· Player selects which level s/he plays

· System starts to game with selected level and spacecraft.

Alternative Event Flow(**s**):

· Player can back to spacecraft selection screen

Use Case #4

Use Case: Move Spacecraft

Participating Actor: Player

Pre-conditions:

· The game must have started

· Player must press the keys which move spacecraft

Post-conditions:

· Spacecraft moves successfully according to pressed key

Main Event Flow:

· Player presses the keys in order to move spacecraft

· System changes the coordinates of spacecraft according to pressed keys

---

Use Case #5

Use Case: Fire

Participating Actor: Player

Pre-conditions:

· The game must have already started

Post-conditions:

· The enemy takes damage or player misses target.

Main Event Flow:

· Player presses the key to fire

· System creates a new object which is moving.

· System gives damage to enemies, if the object crush with enemies

· If the object exceeds the limit of seen map, it disappears and player misses the target.

Alternative Event Flow(**s**):

· Player may miss the target

· System deletes the bullet when it leaves the screen

Use Case #6

Use Case: Use Buff

Participating Actor: Player

Pre-conditions:

· The game must have started

· At least one of the buff slots must have a buff

Post-conditions:

· Player’s spacecraft improved by used buff

Main Event Flow:

· In order to get buff, Player must kill enemy spacecrafts

· System drops buff randomly from enemy spacecraft

· Player collects the buff that is dropped from enemy spacecraft

· System improves spacecraft according to collected buff

Use Case #7

Use Case: Pause Game

Participating Actor: Player

Pre-conditions:

· Player must be playing the game

Post-conditions:

· The game is stopped successfully

Main Event Flow:

· Player presses “Pause” button from the game screen

· System stops the game and return the pause menu to player

Use Case #8

Use Case: Mute Game

Participating Actor: Player

Pre-conditions:

· Player must be in the game

Post-conditions:

· The game is muted successfully

Main Event Flow:

· Player presses the mute button during the game

· System set sound settings

5.2. Dynamic Models

5.2.1. Sequence Diagrams

Scenario: Passing from pre boss game to boss game.

In order to pass the pre boss game, the player should destroy all enemies and stations without being destroyed. We explained the details of passing the primary boss game below.

PreBossGameController class reads the inputs coming from the player. According to the inputs, the main loop of the game is executed by timerPulse(). In the timerPulse() method, checkMapSituation() method of PreBossMapContoller class is called. In order to detect collisions in the pre boss map, checkMapSituation() calls checkCollisions() method for every object privately.

Collisions between enemies and bullets are found by checkCollisions(Enemy, HashMap <Long, Bullet>) method. If any of the bullets crash into an enemy, the enemy's health is decreased according to the damage of the bullet. This procedure is repeated for all the enemies in the pre boss map.

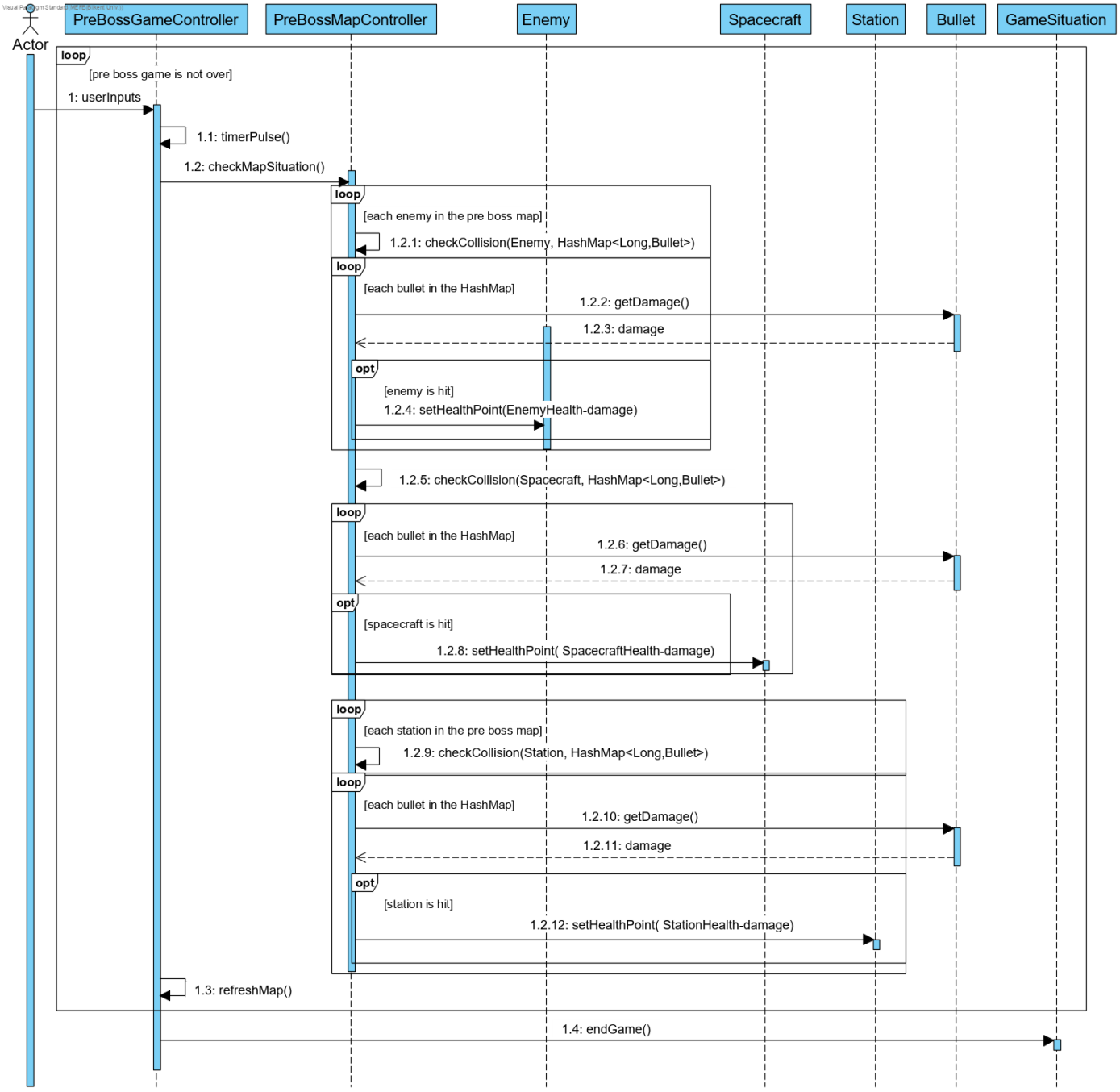
Crashes that spacecraft involves are found by checkCollisons(Spacecraft, HashMap <Long, Bullet>). If any of the bullets hit the spacecraft, spacecraft's health is decreased accordingly.

Lastly, collisions that stations make are found using checkCollisions(Station, HashMap <Long, Bullet>). If any of the bullets hit the station, the station's health point is decreased. This procedure is repeated for every station in the pre boss map

PreBossGameController refreshes the map using refreshMap(). In the refreshMap() method, dead enemies, stations and bullets are removed from the map and spacecraft's health is checked to decide if the pre boss game should continue.

All the actions above are repeated if the pre boss game is not over. However, if the pre boss game is over, GameSituation class is updated. If the pre boss game is finished successfully, the player can face with the boss.

Note that there are other things in the pre boss map that can result in end of the pre boss game. Some examples are collisions between spacecraft-meteor, spacecraft-enemy, enemy-meteor etc. They are not shown because of simplicity reasons.


Actor
PreBossGameController
PreBossMapController
Enemy
Spacecraft
Station
Bullet
GameSituation
loop
[pre boss game is not over]
1: userInputs
1.1: timerPulse()
1.2: checkMapSituation()
loop
[each enemy in the pre boss map]
1.2.1: checkCollision(Enemy, HashMap<Long,Bullet>)
loop
[each bullet in the HashMap]
1.2.2: getDamage()
1.2.3: damage
opt
[enemy is hit]
1.2.4: setHealthPoint(EnemyHealth-damage)
1.2.5: checkCollision(Spacecraft, HashMap<Long,Bullet>)
loop
[each bullet in the HashMap]
1.2.6: getDamage()
1.2.7: damage
opt
[spacecraft is hit]
1.2.8: setHealthPoint(SpacecraftHealth-damage)
loop
[each station in the pre boss map]
1.2.9: checkCollision(Station, HashMap<Long,Bullet>)
loop
[each bullet in the HashMap]
1.2.10: getDamage()
1.2.11: damage
opt
[station is hit]
1.2.12: setHealthPoint(StationHealth-damage)
1.3: refreshMap()
1.4: endGame()

Scenario: The player picks up a buff.

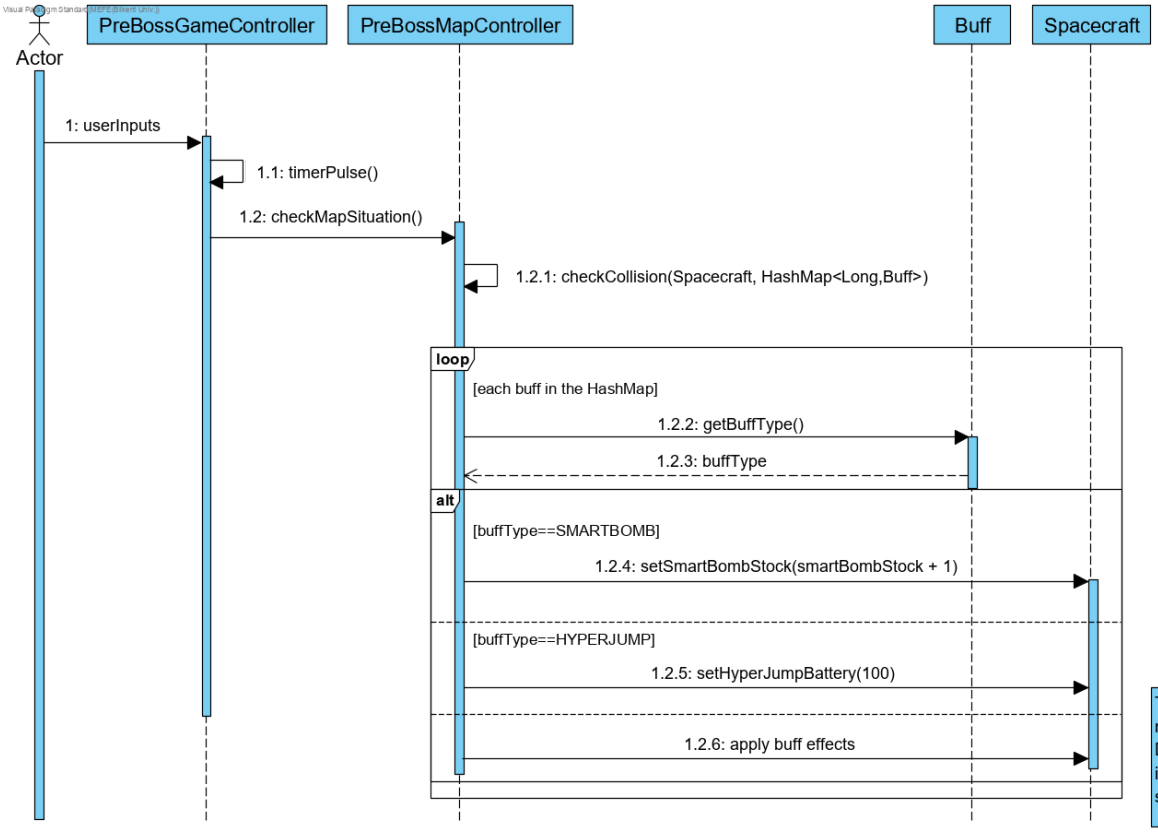
The player wants to pick up a buff and directs his/her spacecraft to the buff using arrow keys. PreBossGameController class reads the inputs coming from the player. According to the inputs, the main loop of the game is executed by timerPulse().

In the timerPulse() method, checkMapSituation() method of PreBossMapController class is called. The method checkMapSituation() calls privately checkCollision(Spacecraft, HashMap<Long,Buff>) method to determine which buff is collided. PrimaryMapController class requests the type of the collided buff. According to type of the collided buff one of the followings happens:

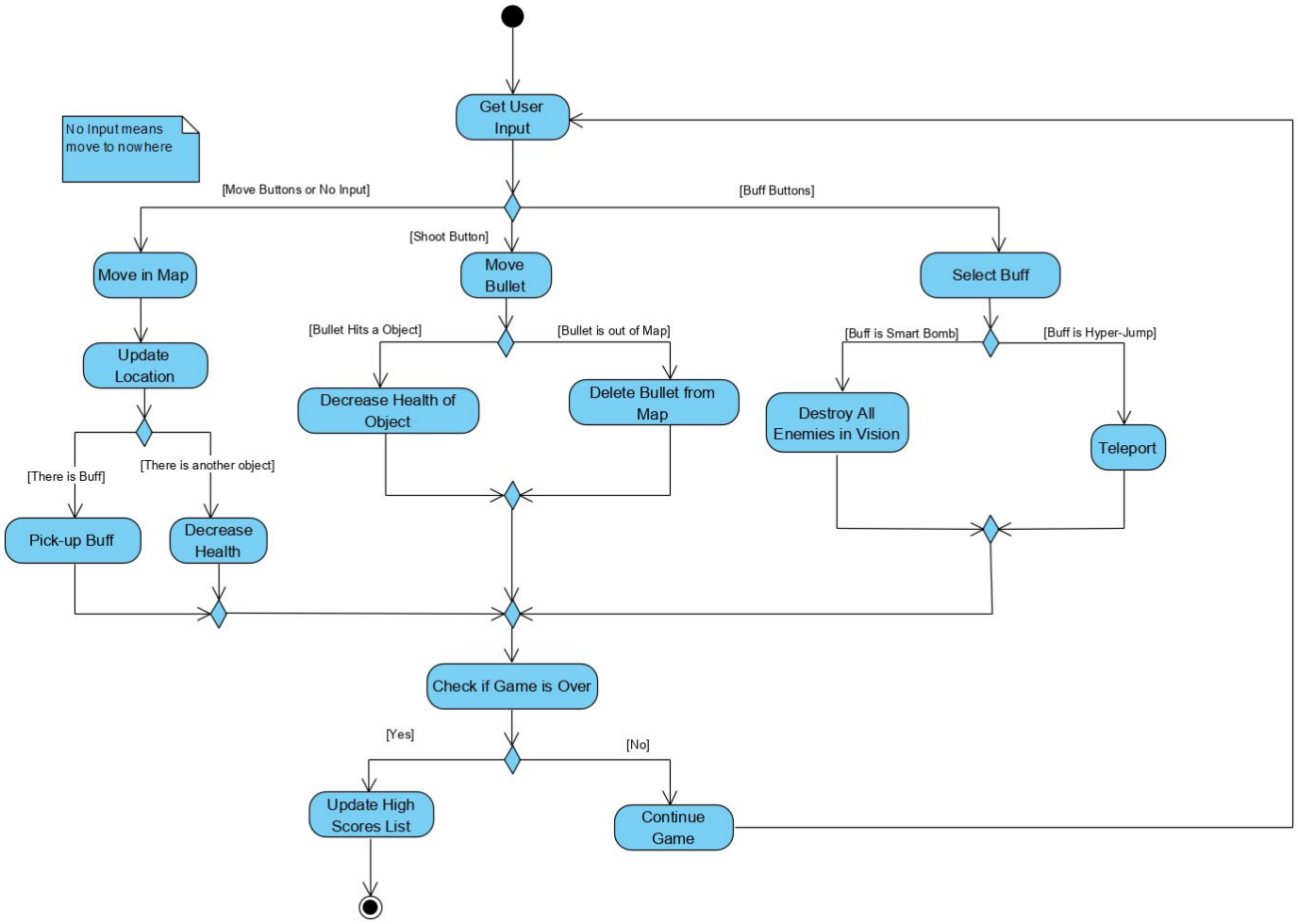
If the type of collided buff is smart bomb, it is added spacecraft's smart bomb stock. It is provided that spacecraft has a missing smart bomb place.

Else if type of collided buff is hyper jump, spacecraft's hyper jump charge is filled up.

Else, collided buff is immediately taken by spacecraft and its effects are applied to spacecraft. For example, if the buff type is health, spacecraft's health point increased.


Actor
PreBossGameController
PreBossMapController
Buff
Spacecraft
1: userInputs
1.1: timerPulse()
1.2: checkMapSituation()
1.2.1: checkCollision(Spacecraft, HashMap<Long,Buff>)
loop
[each buff in the HashMap]
1.2.2: getBuffType()
1.2.3: buffType
alt
[buffType==SMARTBOMB]
1.2.4: setSmartBombStock(smartBombStock + 1)
[buffType==HYPERJUMP]
1.2.5: setHyperJumpBattery(100)
1.2.6: apply buff effects
There is no such method named apply buff effects. Details of the procedure is in the description of the sequence diagram.

5.2.2. Activity Diagram



Given activity diagram shows the general flow of the game and ignores exceptions like pausing the game.

User input is get while game is not over. There are mainly three options the player has next:

Press Move Keys or Do Nothing: Firstly, do nothing means stay at the same location. In other words, it is the base case of moving somewhere. Therefore, it can

be treated as moving. Spacecraft location is changed accordingly to input. If the current location of Spacecraft is same as a buff, the Spacecraft tries to pick it up. If there is another, it crashes the object. The object can be enemy spaceship, enemy station, meteor or enemy bullet. In this case, health of the Spacecraft is decreased.

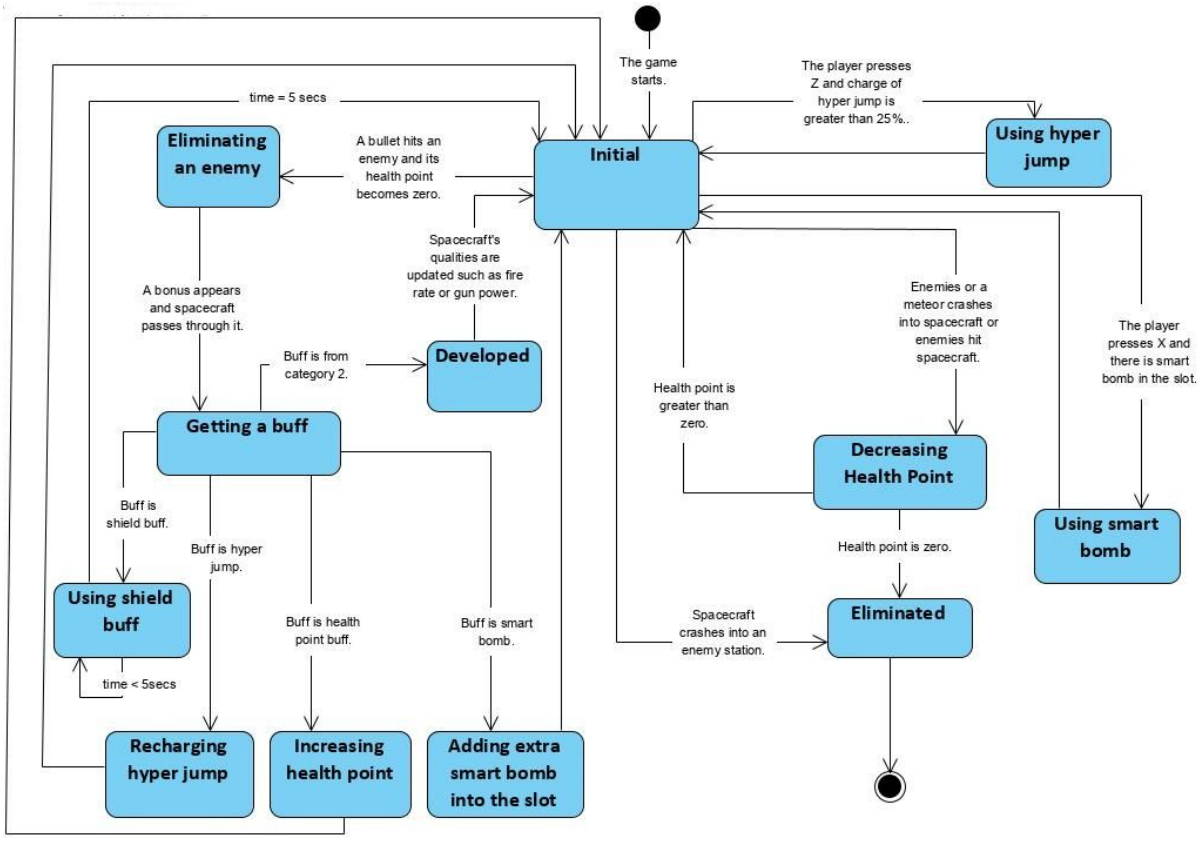
Press Shoot Key: A bullet is sent from spacecraft. If the bullet hits something -again it can be enemy, a station etc. -, object's health is decreased. If the bullet does not hit anything, it leaves the map. After it leaves, the bullet object is destroyed.

Press Buff Buttons: There are two buffs that the player can use and each has a different effect in the game. When the player hits one of the buff keys, type of buff is determined. If the type is smart bomb, enemies that are in the radar of the player are destroyed. If the type is hyper jump, the spacecraft is teleported accordingly.

After acting according to the user input, it is checked whether the game is over or not. If the game is over, scoreboard is updated, and game is end. If it is not over, the game is continuing, and the players next input is waited.

5.3. State Machine Diagrams

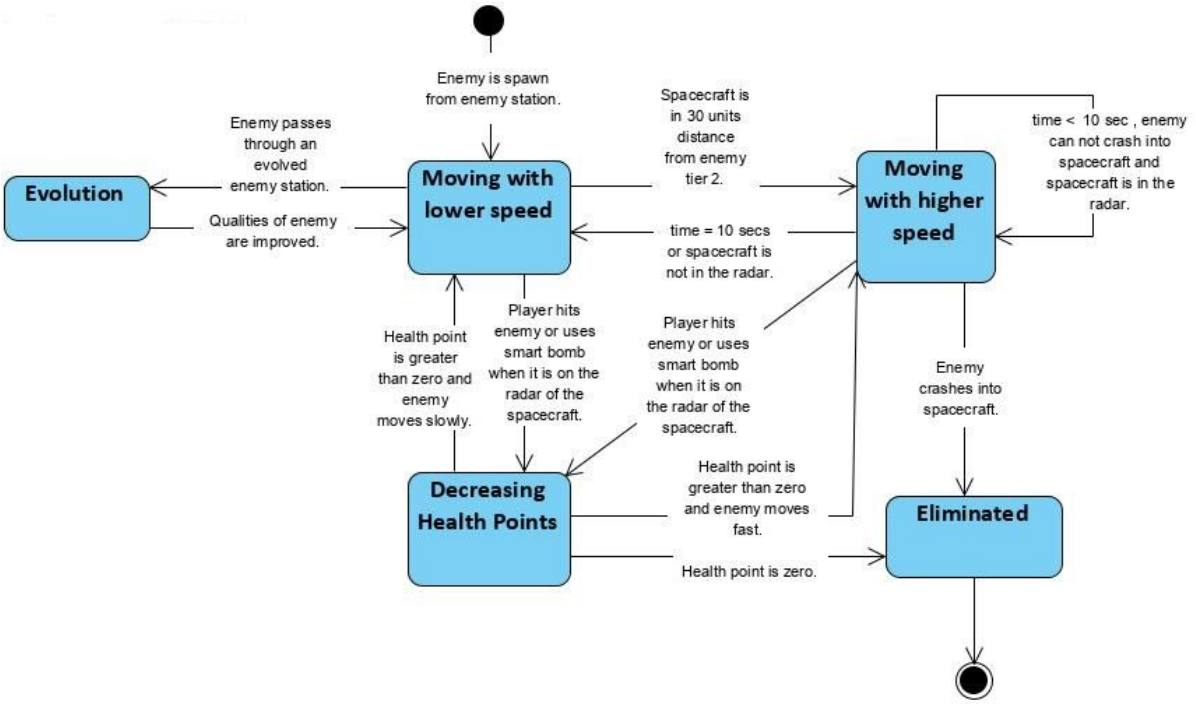
5.3.1. PreBossScene Spacecraft State Machine Diagram



Throughout the game, the spacecraft will have ability to move and fire. However, it will have extra features in different states of its life cycle. The life cycle of the spacecraft will start with the game. When spacecraft fires, it can hit an enemy. If this hit causes the elimination of an enemy, there is a possibility that a buff may appear and the spacecraft takes it. If spacecraft takes a shield buff, buff will be used immediately. It will last 5 seconds and after this, spacecraft will return to the initial state. Namely, it will lose shield ability. If spacecraft takes a hyper jump buff, its charge of hyper jump will be updated to 100%. If it takes health point buff, 15 health points will be added its health point. If it takes smart bomb, spacecraft will have an

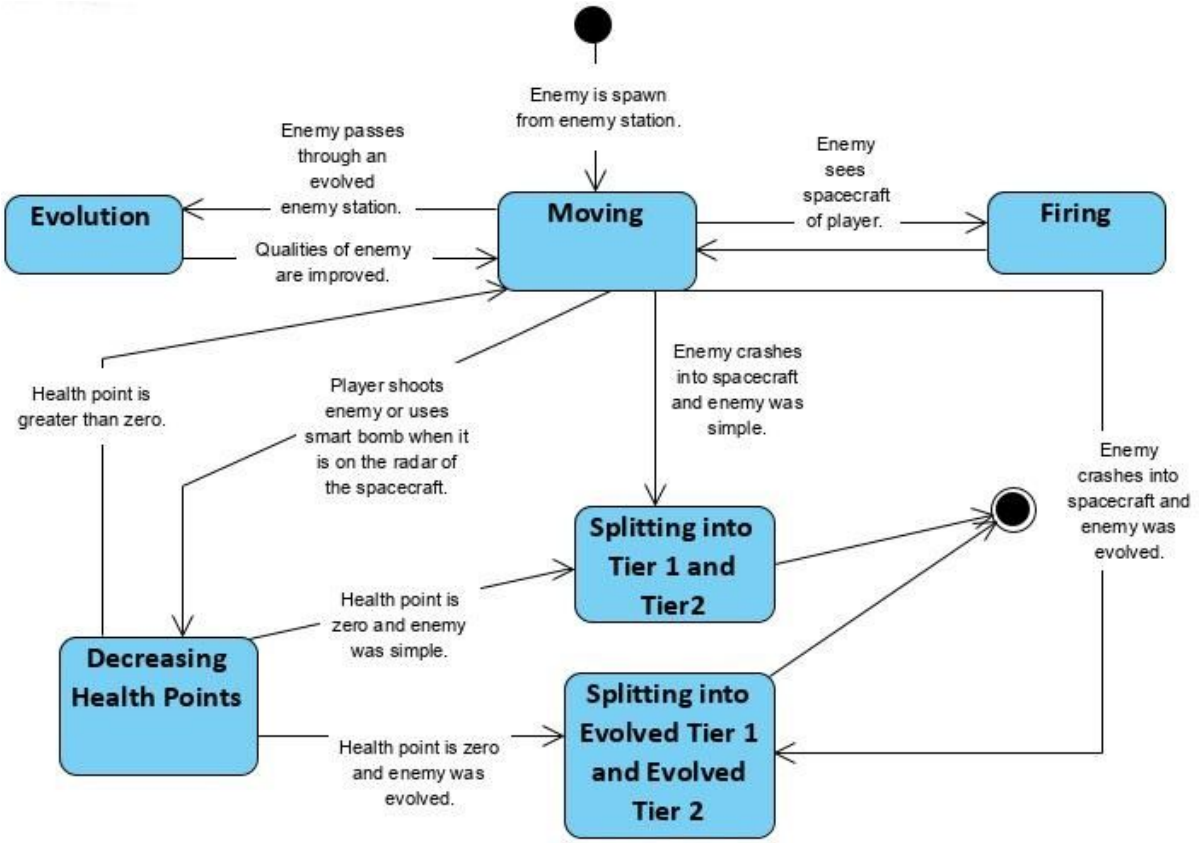
extra smart bomb and it will be added into the slot. If the spacecraft takes a category two buff, its qualities will be improved. Player can use smart bomb or hyper jump abilities by pressing related keys. However, to be able to use these buffs, spacecraft should have at least one smart bomb and 25% charge for hyper jump. If an enemy crashes into spacecraft or it hits spacecraft, spacecraft will lose health points. If spacecraft has health points, it will continue to the game. Otherwise, the life cycle of the spacecraft will end and the game will finish. Spacecraft will also be destroyed when it crashes into an enemy station. This diagram is about the behavior of the spacecraft in PreBossScene. However, its behavior will be similar in the BossScene. In BossScene, buffs will show up when the spacecraft hits the boss of the leve. The player will lose health point when the boss hits spacecraft with the bullet or damages spacecraft using its special abilities. However, if the laser touches spacecraft in level 1, it will be eliminated. In the BossScene, the player can not use hyper jump ability.

5.3.2. Tier 2 Enemy State Machine Diagram



The life cycle of tier 2 enemy starts when it is spawn from enemy station. After it is spawned, tier 2 enemy continuously moves with relatively lower speed. If the spacecraft is on 30 units radar of tier 2 enemy, tier 2 enemy starts to move faster. If it succeeds to crash into spacecraft in 10 seconds, tier 2 enemy will be eliminated. In this process, if spacecraft goes more than 30 units distance from enemy tier 2 or enemy tier 2 can not catch spacecraft in 10 seconds, enemy tier 2 will start to move with lower speed. If the spacecraft hits enemy tier 2, enemy tier 2 will lose health points. If its health point is greater than zero, it will continue to move. However, if its health point is zero, it will be eliminated. Also, if enemy tier 2 is a simple one, it can find an evolved space station and improve its qualities.

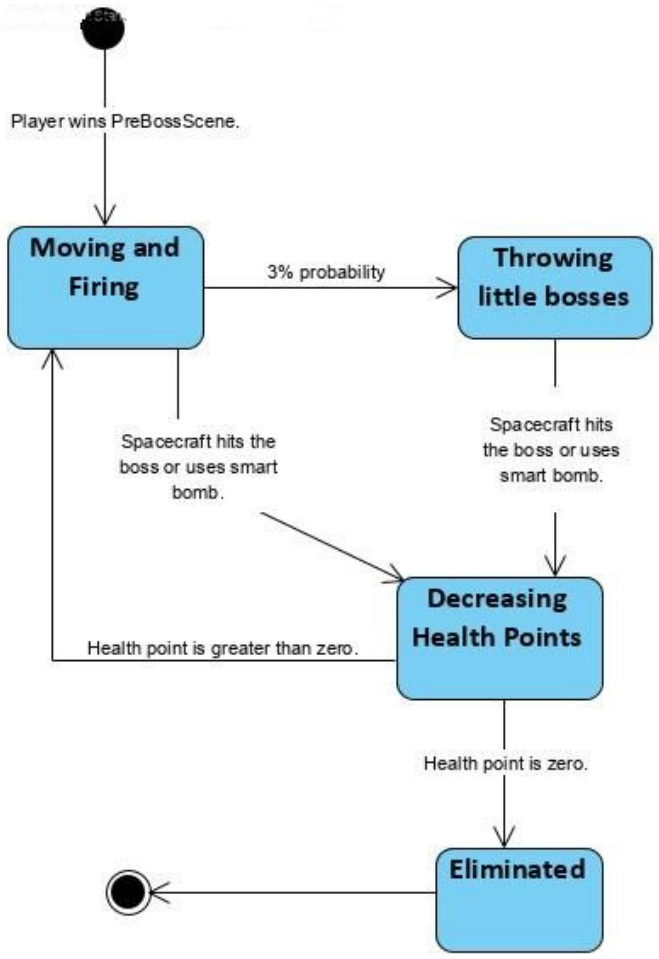
5.3.3. Tier 3 Enemy State Machine Diagram



The life cycle of a tier 3 enemy starts when it is spawned from enemy station. After it is spawned, enemy tier 3 continuously moves until it approaches spacecraft 50 units distance. From this distance, it can see spacecraft and it stops and starts to fire. Also, enemy tier 3 can see a evolved enemy station in 50 units distance. If enemy tier 3 is a simple enemy and it sees a evolved enemy station, it tries to go over it without looking to the position of the spacecraft. After it goes over evolved enemy station, its qualities will be developed and it will continue to move. If the player is able to shoot enemy tier 3 or the player uses smart bomb, enemy tier 3 will lose health points. If its health point is greater than zero, it will continue to move. However, if its health point is zero, it will split into a tier 1 enemy and a tier 2 enemy.

Enemy tier 3 also splits when it crashes into spacecraft. If enemy tier 3 is a simple enemy, it will split into simple enemy tier 1 and simple enemy tier 2. However, if enemy tier 3 is an evolved enemy, it will split into evolved enemy tier 1 and evolved enemy tier 2 and its life cycle will end. Tier 1 enemy has similar behavior with Tier 3 enemy. However, when its health point is zero or it crashes into spacecraft, it is directly eliminated rather than splitting.

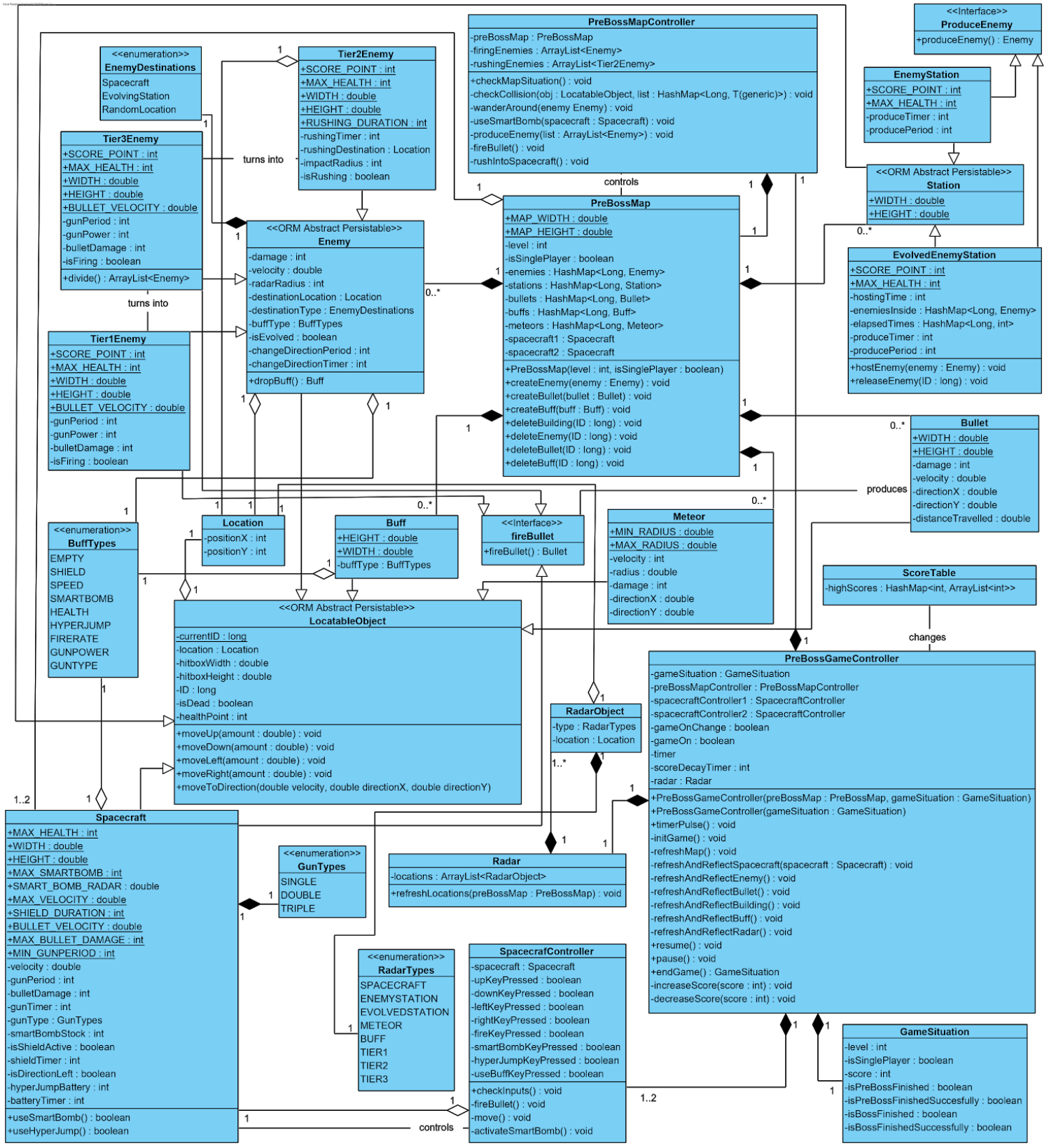
5.3.4. Boss 3 State Machine Diagram

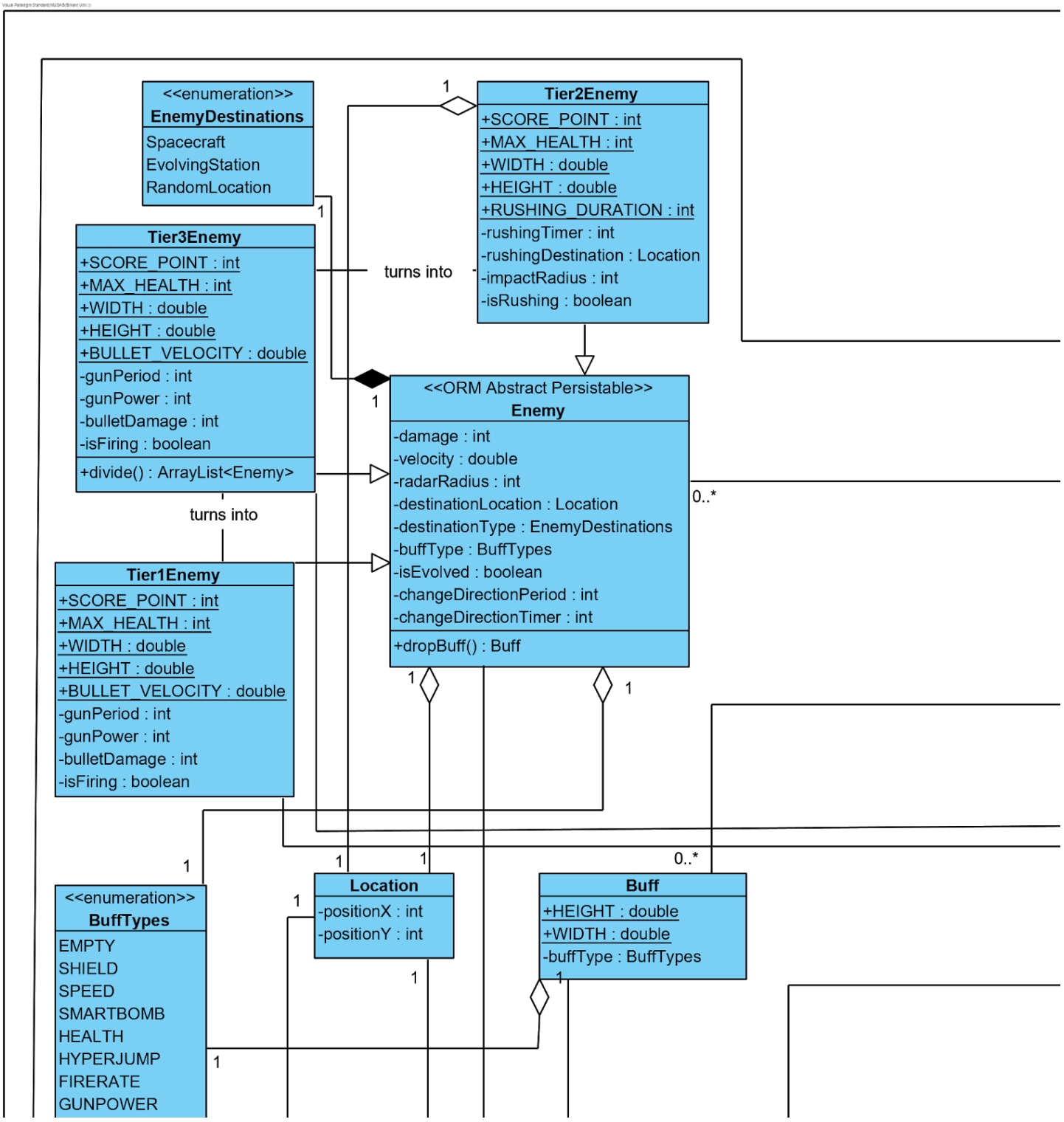


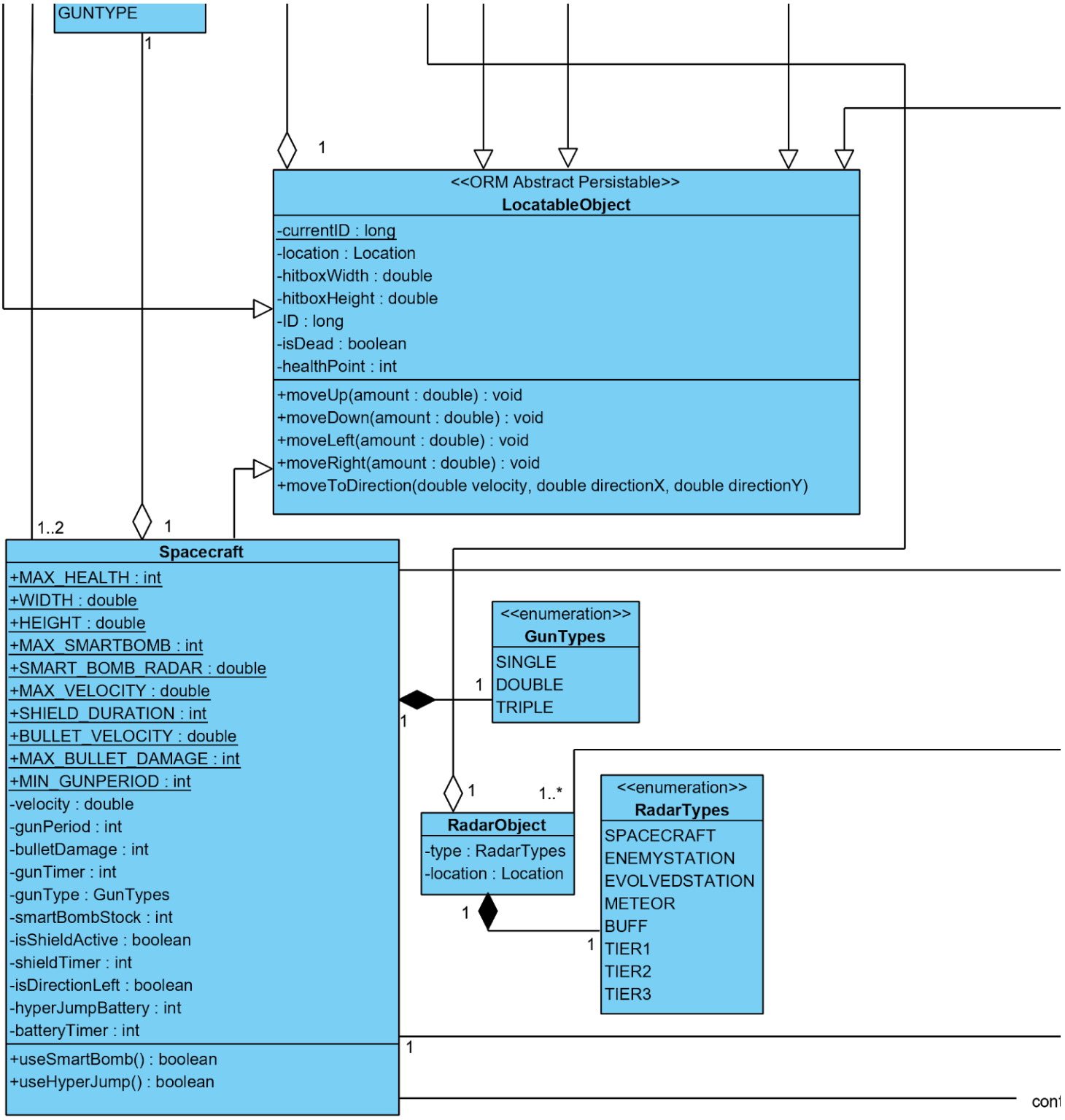
Boss 3 starts its life cycle when the player wins PreBossScene and deserves to fight against Boss 3 in level 3. Boss 3 continuously moves and fires until it throws little bosses with %3 probability. While throwing little bosses, it does not move. If the player hits the boss while it moves or it stops and throws little bosses or the player uses smart bomb, the boss loses health points. If its health point is greater than zero, it can continue to move. However, if its health point is zero, the boss will be eliminated and the level will end. Boss 1 and Boss 2 have also similar behavior. However, their special ability and probability to use these abilities are different from each other.

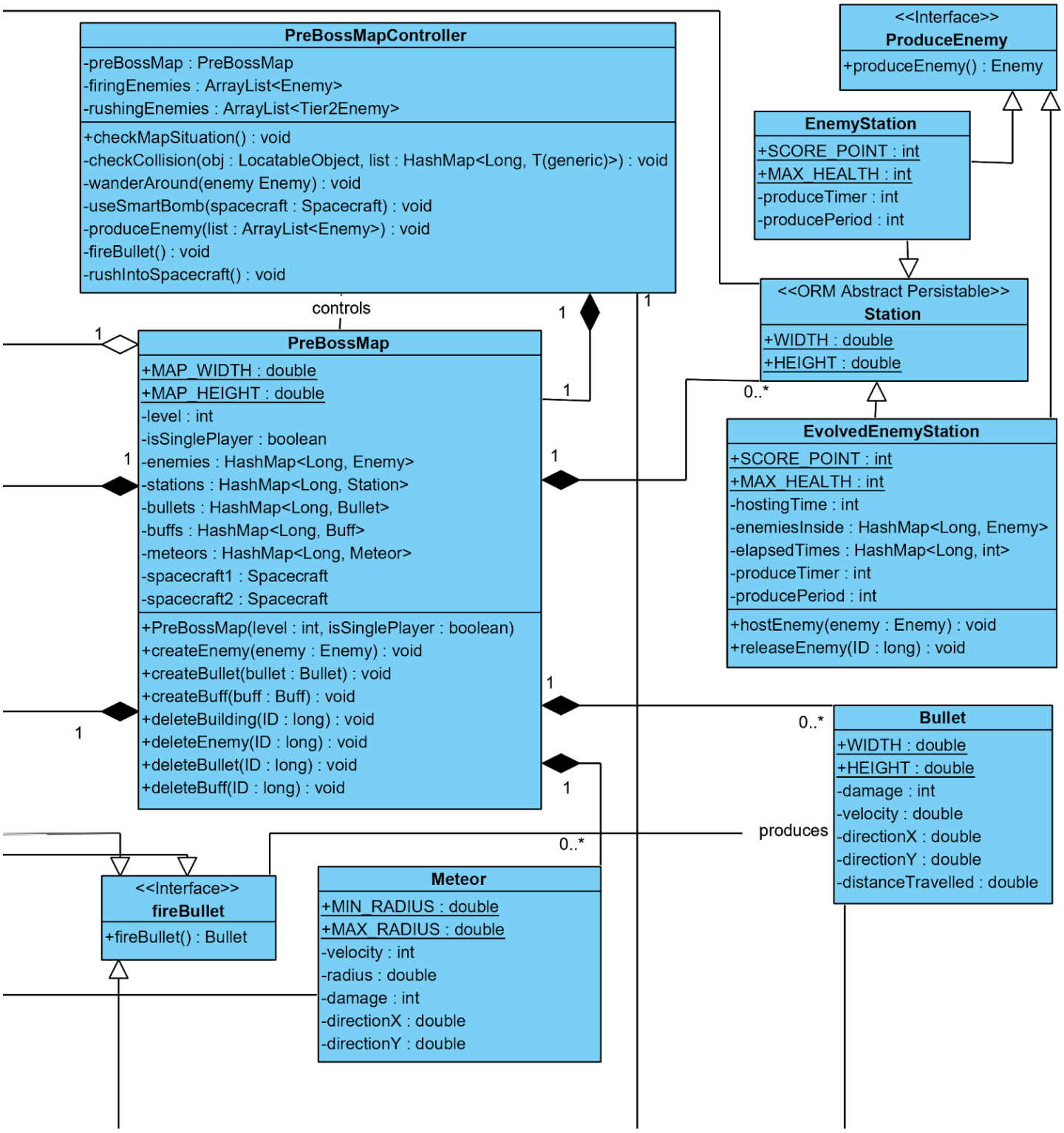
5.4. Object and Class Model

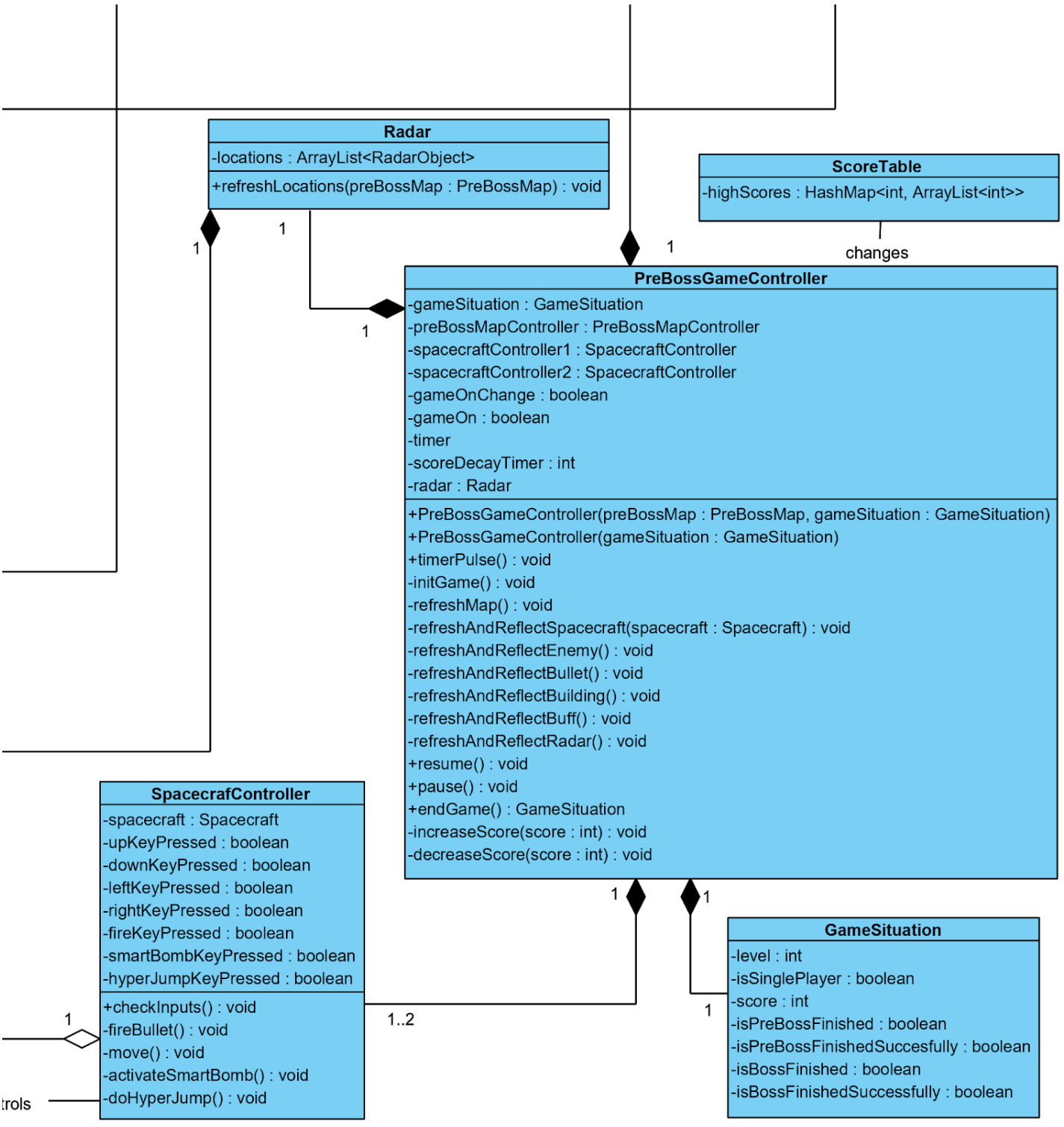
5.4.1. Pre Boss Scene Class Model




<<enumeration>>
EnemyDestinations
Spacecraft
EvolvingStation
RandomLocation
Tier2Enemy
+SCORE_POINT : int
+MAX_HEALTH : int
+WIDTH : double
+HEIGHT : double
+RUSHING_DURATION : int
-rushingTimer : int
-rushingDestination : Location
-impactRadius : int
-isRushing : boolean
Tier3Enemy
+SCORE_POINT : int
+MAX_HEALTH : int
+WIDTH : double
+HEIGHT : double
+BULLET_VELOCITY : double
-gunPeriod : int
-gunPower : int
-bulletDamage : int
-isFiring : boolean
+divide() : ArrayList<Enemy>
turns into
turns into
<<ORM Abstract Persistable>>
Enemy
-damage : int
-velocity : double
-radarRadius : int
-destinationLocation : Location
-destinationType : EnemyDestinations
-buffType : BuffTypes
-isEvolved : boolean
-changeDirectionPeriod : int
-changeDirectionTimer : int
+dropBuff() : Buff
Tier1Enemy
+SCORE_POINT : int
+MAX_HEALTH : int
+WIDTH : double
+HEIGHT : double
+BULLET_VELOCITY : double
-gunPeriod : int
-gunPower : int
-bulletDamage : int
-isFiring : boolean
<<enumeration>>
BuffTypes
EMPTY
SHIELD
SPEED
SMARTBOMB
HEALTH
HYPERJUMP
FIRERATE
GUNPOWER
Location
-positionX : int
-positionY : int
Buff
+HEIGHT : double
+WIDTH : double
-buffType : BuffTypes
1
0..*


GUNTYPE
1
1
<<ORM Abstract Persistable>>
LocatableObject
-currentID : long
-location : Location
-hitboxWidth : double
-hitboxHeight : double
-ID : long
-isDead : boolean
-healthPoint : int
+moveUp(amount : double) : void
+moveDown(amount : double) : void
+moveLeft(amount : double) : void
+moveRight(amount : double) : void
+moveToDirection(double velocity, double directionX, double directionY)
1..2
1
Spacecraft
+MAX_HEALTH : int
+WIDTH : double
+HEIGHT : double
+MAX_SMARTBOMB : int
+SMART_BOMB_RADAR : double
+MAX_VELOCITY : double
+SHIELD_DURATION : int
+BULLET_VELOCITY : double
+MAX_BULLET_DAMAGE : int
+MIN_GUNPERIOD : int
-velocity : double
-gunPeriod : int
-bulletDamage : int
-gunTimer : int
-gunType : GunTypes
-smartBombStock : int
-isShieldActive : boolean
-shieldTimer : int
-isDirectionLeft : boolean
-hyperJumpBattery : int
-batteryTimer : int
+useSmartBomb() : boolean
+useHyperJump() : boolean
<<enumeration>>
GunTypes
SINGLE
DOUBLE
TRIPLE
1
1
1
1..*
RadarObject
-type : RadarTypes
-location : Location
1
1
<<enumeration>>
RadarTypes
SPACECRAFT
ENEMYSTATION
EVOLVEDSTATION
METEOR
BUFF
TIER1
TIER2
TIER3
1
cont


PreBossMapController
-preBossMap : PreBossMap
-firingEnemies : ArrayList<Enemy>
-rushingEnemies : ArrayList<Tier2Enemy>
+checkMapSituation() : void
-checkCollision(obj : LocatableObject, list : HashMap<Long, T(generic)>) : void
-wanderAround(enemy Enemy) : void
-useSmartBomb(spacecraft : Spacecraft) : void
-produceEnemy(list : ArrayList<Enemy>) : void
-fireBullet() : void
-rushIntoSpacecraft() : void
<<Interface>>
ProduceEnemy
+produceEnemy() : Enemy
EnemyStation
+SCORE_POINT : int
+MAX_HEALTH : int
-produceTimer : int
-producePeriod : int
<<ORM Abstract Persistable>>
Station
+WIDTH : double
+HEIGHT : double
controls
1
1
1
1
1
1
0..*
PreBossMap
+MAP_WIDTH : double
+MAP_HEIGHT : double
-level : int
-isSinglePlayer : boolean
-enemies : HashMap<Long, Enemy>
-stations : HashMap<Long, Station>
-bullets : HashMap<Long, Bullet>
-buffs : HashMap<Long, Buff>
-meteors : HashMap<Long, Meteor>
-spacecraft1 : Spacecraft
-spacecraft2 : Spacecraft
+PreBossMap(level : int, isSinglePlayer : boolean)
+createEnemy(enemy : Enemy) : void
+createBullet(bullet : Bullet) : void
+createBuff(buff : Buff) : void
+deleteBuilding(ID : long) : void
+deleteEnemy(ID : long) : void
+deleteBullet(ID : long) : void
+deleteBuff(ID : long) : void
EvolvedEnemyStation
+SCORE_POINT : int
+MAX_HEALTH : int
-hostingTime : int
-enemiesInside : HashMap<Long, Enemy>
-elapsedTimes : HashMap<Long, int>
-produceTimer : int
-producePeriod : int
+hostEnemy(enemy : Enemy) : void
+releaseEnemy(ID : long) : void
1
1
1
0..*
Bullet
+WIDTH : double
+HEIGHT : double
-damage : int
-velocity : double
-directionX : double
-directionY : double
-distanceTravelled : double
produces
0..*
<<Interface>>
fireBullet
+fireBullet() : Bullet
Meteor
+MIN_RADIUS : double
+MAX_RADIUS : double
-velocity : int
-radius : double
-damage : int
-directionX : double
-directionY : double


Radar
-locations : ArrayList<RadarObject>
+refreshLocations(preBossMap : PreBossMap) : void
ScoreTable
-highScores : HashMap<int, ArrayList<int>>
changes
PreBossGameController
-gameSituation : GameSituation
-preBossMapController : PreBossMapController
-spacecraftController1 : SpacecraftController
-spacecraftController2 : SpacecraftController
-gameOnChange : boolean
-gameOn : boolean
-timer
-scoreDecayTimer : int
-radar : Radar
+PreBossGameController(preBossMap : PreBossMap, gameSituation : GameSituation)
+PreBossGameController(gameSituation : GameSituation)
+timerPulse() : void
-initGame() : void
-refreshMap() : void
-refreshAndReflectSpacecraft(spacecraft : Spacecraft) : void
-refreshAndReflectEnemy() : void
-refreshAndReflectBullet() : void
-refreshAndReflectBuilding() : void
-refreshAndReflectBuff() : void
-refreshAndReflectRadar() : void
+resume() : void
+pause() : void
+endGame() : GameSituation
-increaseScore(score : int) : void
-decreaseScore(score : int) : void
SpacecrafController
-spacecraft : Spacecraft
-upKeyPressed : boolean
-downKeyPressed : boolean
-leftKeyPressed : boolean
-rightKeyPressed : boolean
-fireKeyPressed : boolean
-smartBombKeyPressed : boolean
-hyperJumpKeyPressed : boolean
+checkInputs() : void
-fireBullet() : void
-move() : void
-activateSmartBomb() : void
-doHyperJump() : void
GameSituation
-level : int
-isSinglePlayer : boolean
-score : int
-isPreBossFinished : boolean
-isPreBossFinishedSuccesfully : boolean
-isBossFinished : boolean
-isBossFinishedSuccessfully : boolean
1
1..2

Fiyuv++'s pre boss scene object class diagrams are shown in the figures above. First figure is the overview and other figures are split versions of the first figure. These classes are focused only on application domain. In other words, graphical user interface, menu, settings and other class diagrams related to solution domain are not shown in these diagrams.

Interaction between classes are shown in the figures, nevertheless, more explained versions of these classes such as their purposes and their functions' explanations except getters, setters can be found below. Firstly, three major classes PreBossGameController, PreBossMapController, and SpacecraftController classes are explained. These classes control the pre boss game scene in general and with help of abstraction one can partially understand how the pre boss game scene works. After these classes have been explained, entity objects of the game are beginning from LocatableObject Class since any object on the pre boss map inherits LocatableObject. The subclasses of LocatableObject will have specifically initialized width and height attributes. In order to avoid from repetition these attributes are only mentioned in here. Lastly, rest of the entity classes are explained.

5.4.1.1. PreBossGameController and GameSituation Class

This class has the overall control of the pre boss game scene as it can be understood by its name. It deals with the user input with the help of SpacecraftController class and creates timer pulses in order the scene to process to its next stage. It deals with the game flow with the help of PreBossMapController class by creating timer pulses. It has the current situation of the game such as is it a

single player or multiplayer version; which level this is; whether this scene has ended and if it ended whether it was successful or not; or what is the current score. There is a Radar attribute to hold current radar according to the map. This class builds the interaction between spacecraft related user input to SpacecraftController class. For the other inputs such as pausing the game, there are attributes such as gameOn and gameOnChange in order to pass between the different states of the game resumed or paused.

As it can be understood GameSituation, Radar, SpacecraftController, PreBossMapController have no life-cycles except in this class; therefore, there are composition relationships between these classes and PreBossGameController class.

PreBossGameController class will get initial game situation as parameter in the constructor method and according to level value it will design the beginning of game with private “initMap” method. For example, if the level is two, there will be randomly created tier 1 and tier 2 enemies on the map, but no tier 3 enemies. After game starts, PreBossGameController class will control over the stages of game such as pause stage, resume stage, refreshing map, and finally ending the pre boss scene. In the normal flow of the game, which is not in pause, this class will constantly call refresh map method to inform PreBossMapContoller to process PreBossMap to its next stage according to current situation. The class will change scores and radar constantly according to the flow of the game.

5.4.1.2. SpacecraftController Class

This class has the control over a spacecraft object of the map. If the game is two player version, there will be two objects of this controller class for each of the spacecrafts. These class have attributes for spacecraft and which key is currently pressed by the user such as movement keys, using buff key, using smart bomb key, using hyper jump key.

These classes have aggregation relationship with Spacecraft class since it is a part of SpacecraftController class.

This class has one superior method “checkInputs” that calls other methods privately if they are needed. This method is called on every timer pulse by PreBossGameController class. There are methods for firing bullet, moving, using buffs such as normal buffs, smartbomb, hyper jump. These methods are called according to the user input.

5.4.1.3. PreBossMapController Class

This class has the control over PreBossMap. It holds PreBossMap as attribute in this class and constantly manipulates according to flow of the game with the help of PreBossGameController class. In other words, PreBossGameController class provides the constant timer pulses for PreBossMapController class.

Without PreBossMapController class, an object of PreBossMap class wouldn’t be meaningful, therefore, there is a composition relationship between these classes.

The most important methods of PreBossMapController are “checkMapSituation” and “checkCollision”. These two classes mainly checks every object’s situation to other objects on the map at every timer pulse. According to this check, necessary actions are taken and other methods are called from “checkMapSituation” as well. For example, if a bullet and an enemy intersect at the current timer pulse, the bullet is marked as dead to be removed from the map and the enemy’s health point is decreased according to the bullet’s damage or even the enemy can be marked as dead if it’s health point is less than 1.

5.4.1.4. LocatableObject (Abstract)

This class is a simple but effective class. It is important to note that this class is an abstract class meaning that there will be no objects of this class except manipulating via polymorphism. This class represents any object displayed on the map for both pre boss game scene and boss scene. Hence, there will be a lot of classes that will inherit this class. According to this requirement there are 5 important attributes of this class which are location (x, y coordinate), hitbox sizes(width and height), unique ID number, health point, and isDead attribute to check whether the object’s life-cycle on the map has ended. Some subclasses will have specific health point boundaries such as 100 for Spacecraft, and some subclasses won’t have specific health point boundaries which will lead them to have 1 health point when they created. To exemplify, Bullet or Meteor will automatically have 1 health point and when they clash to other objects on the map they will be dead.

Location is another class, but since it only consists of X coordinate and Y coordinate there is no need for further explanation. LocatableObject class has this class as attribute but without this class there could be other Location objects as well, therefore, there is an aggregation relationship between these classes.

Important LocatableObject class methods other than attribute manipulation methods are movement methods which change the coordinate of the object via Location class.

5.4.1.5. PreBossMap Class

This class is like a real-life like map. It represents the current situation of the pre boss game scene. The representation is provided by holding stations, enemies, bullets, meteors, or users spacecrafts as list attributes.

PreBossMap class has the lists of entity classes such as Station, Spacecraft, Bullet, Enemy, Meteor, Buff as attributes and these classes don’t have any life-cycles except being attributes of PreBossMap class, thus, there are composition relationships between PreBossMap class and other classes mentioned.

Current situation of the map will be constantly changing according to the flow of the game such as producing enemies from enemy stations or randomly generated meteors. As mentioned in PreBossGameController class there will be a level parameter to initialize the game according to the current level. PreBossMap will be start according to this parameter as well.

5.4.1.6. Enemy (Abstract)

This class is an abstract class to represent common features of simple enemies of different tiers. It has informative attributes of an enemy such as health, damage, velocity, radar, buff(optional), evolving situation and destination. These attributes are common for simple enemies. Evolution of enemies are done by changing its attributes. This class has “dropBuff” method to create a buff on the map when the enemy is dead in case there is a buff in the dead enemy. Other than this method there are attribute manipulation methods.

This class inherits LocatableObject class, has aggregation relationship with Location class to represent destination location, has composition relationship with PreBossMap class.

There are 3 different classes that inherits this class which are enemies of Tier 1,2,3.

5.4.1.7. Tier1Enemy, Tier2Enemy, Tier3Enemy

These classes are representation of enemies according to their tiers. Tier1 enemies can be seen in level 1, 2, 3. Tier2 enemies can be seen in level 2, 3. Lastly, Tier3 enemies can only be seen at level 3. These classes specific boundaries considering their score points they will give when they die or their creation time health points.

Tier1 and Tier3 enemies shot bullets via FireBullet interface and have related attributes to shot bullet. Tier2 enemies are suicide planes, so they have different

attributes related to rushing into a spacecraft. Tier3 enemies can divide into Tier1 and Tier2 enemies with “divide” method. Lastly, since this classes inherit Enemy class, evolution of enemies within their tier is done by changing their attributes from Enemy class and their own class together.

5.4.1.8. Spacecraft

This class represents the players on the map. It has final static attributes for boundaries such as max health, smart bomb capacity, max velocity, shield duration, bullet velocity, max bullet damage. It has other non-static attributes for gun features(power, period, type), velocity, buff slot, smart bomb magazine, battery for hyper jump, infinite armor buff, timers to control time related features such as battery increase in time or shield duration. Spacecraft can fire bullets via fireBullet interface as tier 1, 3 enemies.

This class has methods for using buffs such as infinite armor for a limited time, increase spacecraft’s gun features or its velocity. There are also methods to use smart bombs and hyper jums. Other methods are to manipulate rest of the attributes.

This class inherits LocatableObject class and has composition relationship with PreBossMap class.

5.4.1.9. Bullet

This class is a small class to represent bullets that are produced by fireBullet interface. Bullets have damage, velocity, and direction attributes, and distance travelled. This class has only methods to manipulate these attributes.

This class inherits LocatableObject class and has composition relationship with Map class.

5.4.1.10. Station(Abstract)

This is a simple class to represent enemy stations on the map.

This class inherits LocatableObject class. Different stations such as EnemyStation and EvolvedEnemyStation inherit this class. This class has a composition relationship with PreBossMap class.

5.4.1.11. EnemyStation, EvolvedEnemyStation

These classes are specialized station classes for different types of enemy stations. These classes have specific health and score boundaries. EvolvedEnemyStation hosts unevolved simple enemies to evolve them. EnemyStation and EvolvedEnemyStations produce unevolved and evolved simple enemies respectively by inheriting ProduceEnemy interface.

5.4.1.12. Meteor

This is a simple class to represent meteors on the map. Meteors has different sizes and different directions. This class has direction, velocity, damage attributes. Velocity and damage are changing according to the size of meteor.

This class inherits LocatableObject class which controls the size of meteor and has composition relationship with Map class.

5.4.1.13. Buff

This is a simple class to represent buffs on the map. It has type attribute to determine which buff this is. The object of this class is created when a simple enemy is dead, but at a random chance.

This class inherits LocatableObject class and has composition relationship with Map class.

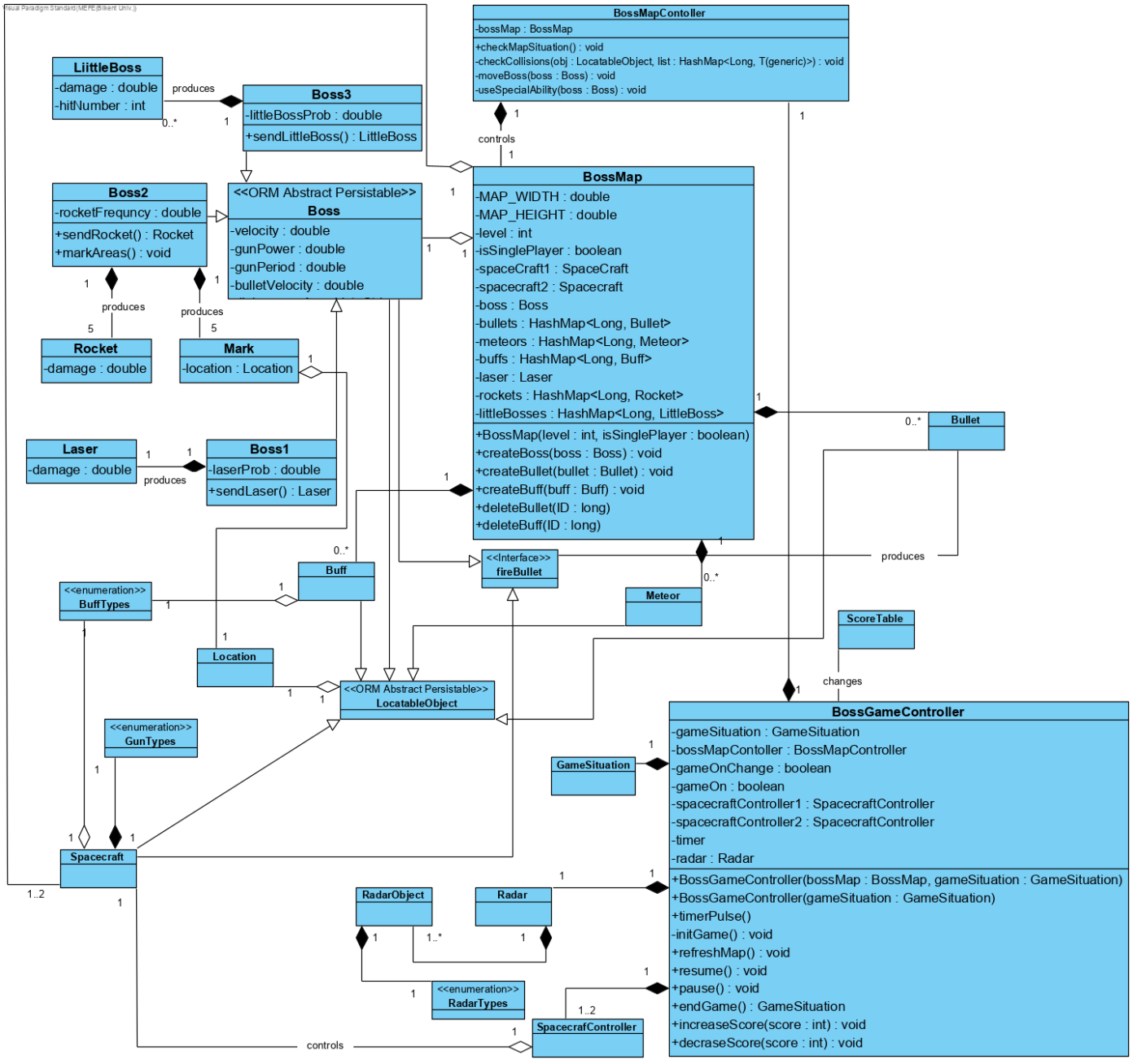
5.4.1.14. Radar

This class represents the current situation of the map with RadarObject class objects containing Location, and types. SpacecraftController class has this class as attribute and manipulates it according to PreBossMap class. Hence, there is a composition relationship between Radar and PreBossGameController class.

5.4.1.15. ScoreTable

This class holds the top 10 high scores for different levels as lists. This class has possibility to be changed by PreBossGameController and BossSceneController class. For one player and two players versions score tables will be different.

5.4.2. Boss Scene Class Model



Boss scene classes use some of the classes explained in section 5.4.1. such as LocatableObject or Spacecraft. In order not to repeat same classes, same classes are shown in the model without showing their attributes and methods. Classes related to boss scene explained below in detail.

5.4.2.1. BossSceneController

Like in PreBossSceneController class, the class controls the general flow of the game. The reason why we create a class similar to PreBossMapController is that this class controls the behaviours of the bosses rather than enemies and stations. The class also has control of pausing game, changing score like PreBossController class does.

5.4.2.2. BossMapController

This class uses time pulses generated by BossSceneController to manipulate BossMap class. Capabilities of the class are following: Its checkCollision method, reveals the collisions happening in the BossMap and stimulates the BossMap class. Likewise, useSpecialAbility method makes the boss in the BossMap use its special ability.

There is a composition relationship between BossSceneController and BossMapController because without the generated pulses, this class is not capable of doing something meaningful.

5.4.2.3. BossMap

The class contains objects from classes Spacecraft,Boss, Meteor, Bullet and one of the Laser, Radar and LittleBoss. Therefore, there is aggregation relationship with

those classes. The class can create or delete various objects according to the directions of BossMapController class.

5.4.2.4. Boss (Abstract)

This is a simple abstract class which represents common properties and methods of 3 different type of bosses. Every boss has a velocity and a regular gun which has a specific power, period, and bullet velocity. This abstract class includes these properties and their getter and setter methods.

This class inherits from LocatableObject class and implements FireBullet interface.

5.4.2.5. Boss1

Boss1 class represents the Boss of level 1. Since the special power of Boss1 is laser, this class includes laserProb attribute to hold the probability of sending laser and it has sendLaser method.

This class inherits Boss abstract class.

5.4.2.6. Laser

This class represents the laser entity. It includes damage attribute to hold the damage of the laser. Laser object is produced by Boss1.

5.4.2.7. Boss2

Boss2 class represents the Boss of level 2. Since the special power of Boss2 is rocket, this class includes rocketFrequency attribute to hold the frequency of sending rocket. This class also has markAreas and sendRocket methods. markAreas method puts a sign to 5 different areas which are determined randomly. Then, sendRocket methods sends rocket to these determined areas.

This class inherits Boss abstract class.

5.4.2.8. Mark

This class represents mark which is marked by Boss2. It has only location attribute.

5.4.2.9. Rocket

This class represents the rocket entity. It includes damage attribute to hold the damage of the rocket. Rocket object is produced by Boss2.

5.4.2.10. Boss3

Boss3 class represents the Boss of level 3. Since the special power of Boss3 is little boss, this class includes littleBossProb attribute to hold the probability of sending little boss. Boss3 class also has sendingLittleBoss method.

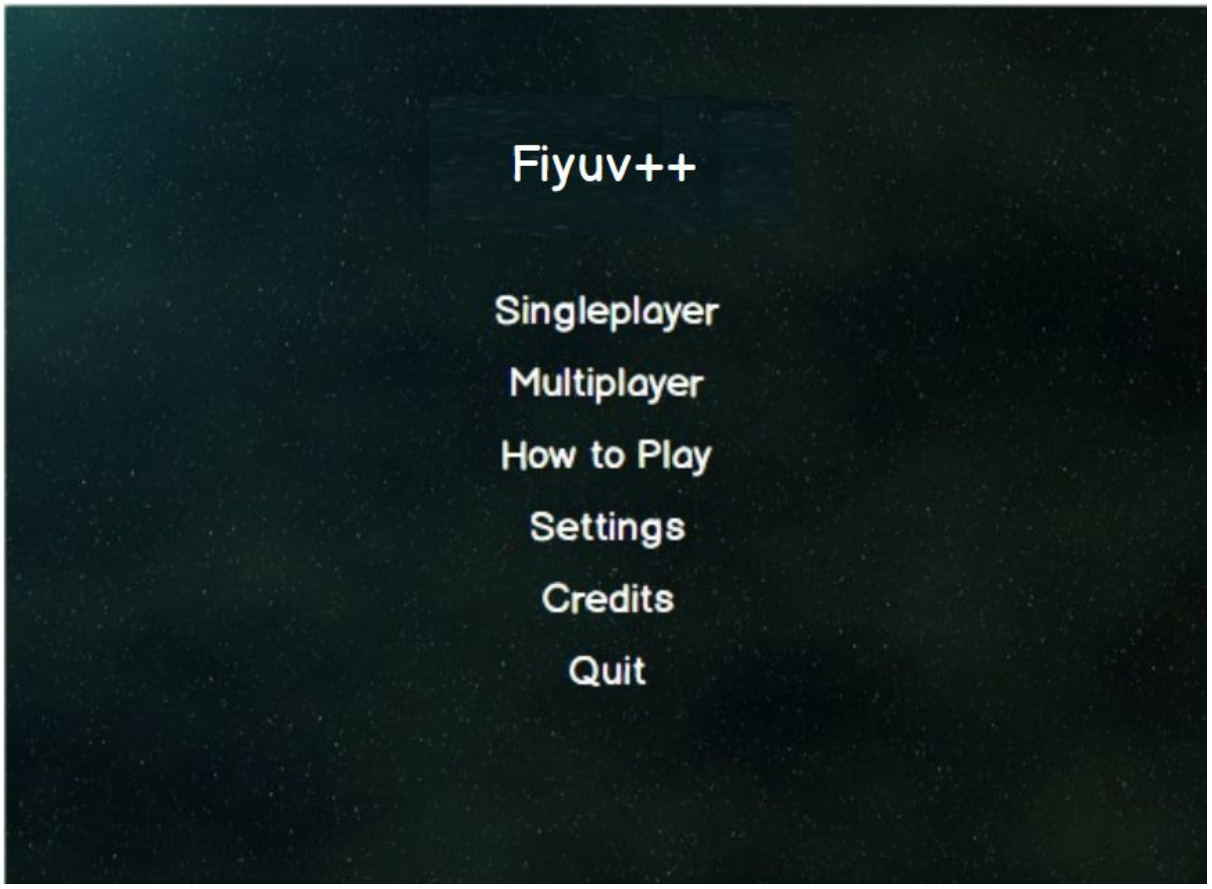
This class inherits Boss abstract class.

5.4.2.11. LittleBoss

This class represents the little boss entity which is produced by Boss3. It includes damage attribute to hold the damage of the little boss. Also, there is another attribute, hitNumber, to hold how many times a little boss hits the boundaries. It is necessary because a little boss disappears after it hits 3 times.

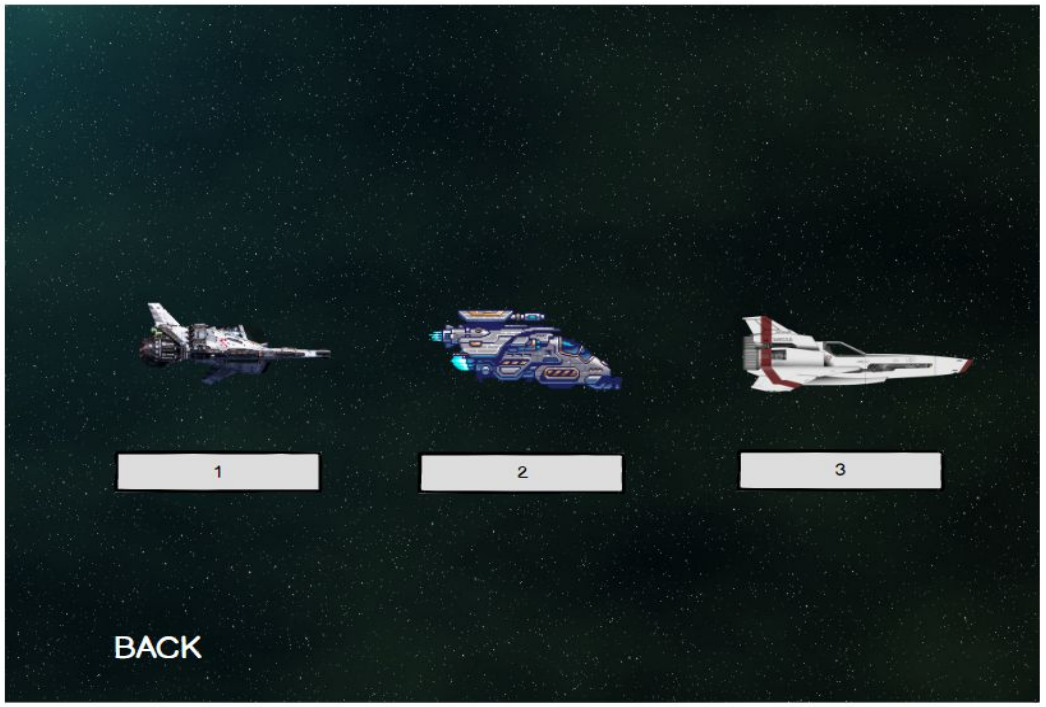
5.5. Screen Mock-ups

5.5.1. Main Menu



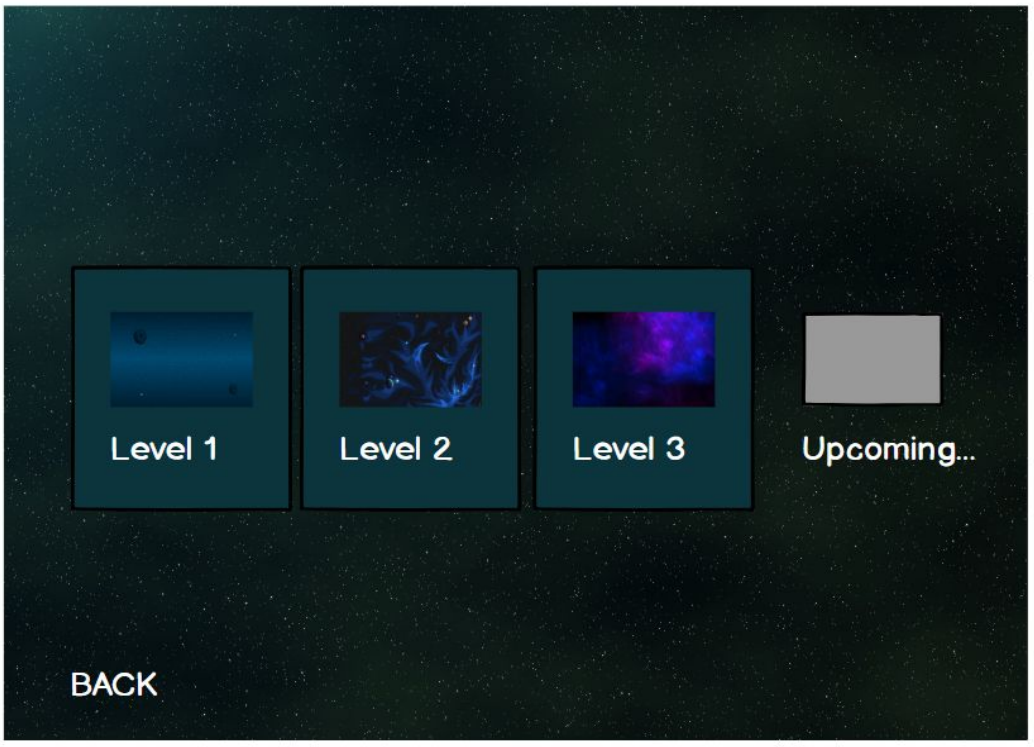
Main menu is the first screen of the game that contains 6 buttons. “Single Player” starts the game with one spaceship while “Two Player” starts the game with two spaceships. “How to Play” provides information about the game. “Settings” enables player to arrange volume and choose which keys will be used in the game. “Credits” provides information about the game developers. If the player wants to exit, “Quit” button will help.

5.5.2. Spaceship Options



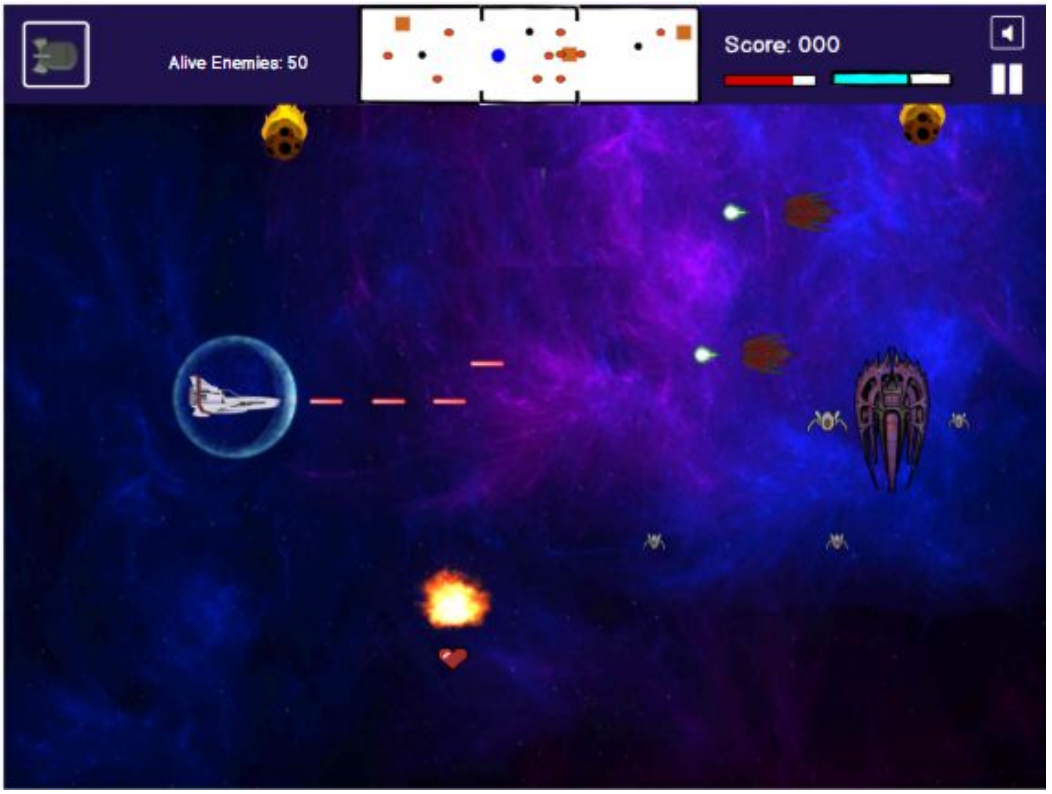
This screen helps the player to choose which spacecraft s/he wants to use in the game. This screen contains a “Back” button to return main menu.

5.5.3. Level Options



The player can choose one of unlocked levels to play from this menu. Although menu shows all levels, the player cannot choose a locked level. The player can use “Back” button turn back to Main Menu.

5.5.4. Map



The map contains enemy space stations, alliance space stations, enemies and player's spaceship. On the left top corner of the map, there are number of alive enemies and a slot for smart bomb. A radar shows the place of enemies and enemy space stations. On the right top corner, there are a button to turn on or off the sound and a button to pause the game, score information, health bar and hyper jump bar.

6. Conclusion

In this report, we explained our analysis about "Defender" game. In "Overview" and "Functional Requirements" parts, we tried to describe basic concepts of our game and clarify which functionalities our game will contain. In "Nonfunctional Requirements" part, we explained how we can increase performance of our game, we discussed portability and extendibility of our game. At the end, we tried to make our analysis more understandable by using system models. In our system models, we included use case diagram to portray functionalities which player can perform, activity diagram to describe behavior of the system, sequence diagrams to explain dynamic behavior between objects of the system, state diagrams to clarify the behavior of various objects, class diagrams to explain static structure of the game, and screen mockups to visualize appearance of the game.

This report is prepared to define all functional and nonfunctional requirements of our "Defender" game. We tried to explain everything clearly. Therefore, we can use this report as a guideline to design and implement our game.

7. Improvement Summary

In the second iteration of the analysis report, we have changed following parts:

1. We dealt with uncertain details such as gun power, health point or speed. We updated all uncertain amounts to exact numbers.
2. We handled inconsistency in the usage of notions and between diagrams.
3. We added state machine diagrams of the spacecraft, tier 2 enemy, tier 3 enemy and boss 3.
4. We changed use case diagram in order to make it consistent.
5. We improved the sequence diagrams and activity diagram so that they can be consistent with other diagrams.
6. We removed ally buildings.
7. We divided our game into two scenes which are pre boss scene and boss scene and adjust our diagrams according to this.
8. We added class diagram for boss scene.

8. Glossary & References

[1] C. Castillo, "JavaFX Architecture," Oracle, 2013. [Online]. Available: https://docs.oracle.com/javafx/2/architecture/jfxpub-architecture.htm.

[2] T. Lindholm, F. Yellin, G. Bracha and A. Buckley, "The Java Virtual Machine Specification," Oracle, California, 2013.

[3] “Defender (1981 video game),” *Wikipedia*, 09-Nov-2019. [Online]. Available: https://en.wikipedia.org/wiki/Defender_(1981_video_game). [Accessed: 25-Nov-2019].