Bilkent University

Department of Computer Engineering

CS 319 Term Project

Defender: Version Fiyuv++

Analysis Report

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: Version Fiyuv++

1. Introduction

Defender is an old arcade game whose first version was released by Williams Electronics in 1981. 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 make people try our new version of Defender. Therefore, we chose Defender as our term project game.

Our version of Defender will have following features:

- Single Player Mode
- Multiplayer Mode
- Sound Options
- Spaceship Options
- Background Color Options
- Different Backgrounds in Different Levels
- Levels with Increasing Difficulty
- Various Enemies with Different Characteristics
- Bonuses
    1. Increasing Speed
    2. Smart Bombs
    3. Freezing Enemies
    4. Increasing Power of Attack
    5. Increasing Fire Rate
    6. Multiple Shot in Different Directions

7. Hyper Jump
8. Shield
9. Health Bonus

- A Radar Displaying the Place of Enemies
- Commanders at the End of Levels

A player will be able to control a spaceship in order to move, attack enemies which are spreading from enemy space stations, and eliminate their space station. The main purpose is to protect humans living in alliance space stations from enemies that are trying to destroy these space stations. At 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 space stations, the player has to defeat commander of the level which shows up at the end of each level. Commanders will have different qualities in each level.

Our game is keyboard controlled. The player will have ability to change keys of the game from settings. The game will also have mode for two players. In this mode, two players will fight against aliens 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

Defender is a battle game which has a story passing in the space. The player controls his/her own spacecraft and fights against a terrifying leader of enemy forces. To face with the leader, the player has to finish different levels successfully. Each level consists of enemy spaceships, enemy space stations where enemy spaceships can spawn, alliance stations, and commander of the level. Aim of the player is to defeat the level’s commander after he/she gets rid of all enemy spaceships and space stations. While the player is fighting against enemy forces, he/she should try to protect alliance stations. If alliance stations are destroyed by the enemy forces, the player loses points. Likewise, when the player demolishes enemy forces, he/she gains points. The player should do all of them as soon as possible because every passing time causes loss of points. The player can take advantage of different bonuses which are obtained from destruction of enemy spaceships. After all enemy forces are destroyed, the player encounters the level's commander to finish the level.

2.2 Map

First of all, there are player's spacecraft and enemies' spaceships. There are also space stations in various places of map. Some of them belong to enemy forces and others belong to the allies. Lastly, meteor showers will take place in the map.

2.3 Initial Properties of Player's Spaceship

The spaceship has followings at the beginning of the game:

- 100% health point which can be decreased due to attack of enemy spaceships, commanders and hit of meteors.
- A gun which has definite power, frequency and single shot ability. Bullets coming from the gun has constant speed.
- Variable velocity with an upper limit.
- Hyper-jump ability which teleports the spaceship to some distance through its direction. The hyper-jump ability is chargeable, and the spaceship has one at the beginning.
- 3 smart bombs which destroy everything in the screen.
- A radar which provides the player to detect enemy forces.
- An empty slot for an extra ability. While playing the game, the player can gain an ability to put it in the slot.

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 Bonuses

There will be several bonuses which can be gained after demolishing enemy spaceships.

- Hyper-jump Charge: It causes to fully recharge the hyper-jump ability.
- Extra Smart Bomb: The player’s spaceship gains extra smart bomb. If player have already 3 smart bombs at that time, he/she can’t take any more.

For other bonuses, the player has one slot. If the slot is empty, the player can take the bonus to user later. Otherwise, he/she can’t take any bonus. The followings are the bonuses which the player can take the slot:

- Health Bonus: It provides the spaceship to increase health point.
- Shield Ability: The spaceship is covered by a shield which destroys enemy spaceships when they touch the shield. Shield will disappear after a while.
- Freeze Ability: Everything in the map – except the player- freeze for a certain amount of time.
- Speed Bonus: It allows player’s spaceship to move faster.
- Multiple Shot Ability: The player’s spaceship gains the ability of shooting different directions at once.
- Increased Attack Speed: The player’s spaceship gains increased fire rate.
- Increased Attack Power: The player’s spaceship deals more damage to enemy forces.
- Increased Bullet Velocity: Bullets coming from the gun travels much more distance in the same time.

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.

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.

Tier 1: Enemies in this tier are basics. They are easy to kill compared to other types. They do not harm player’s spacecraft until they see it with their limited vision.

Tier 2: Enemies in this tier are like suicide plans. They use their high speed to crash into player’s ship. They have the highest damage capability among all types.

Tier 3: These enemies fight against player from a distance by using their shooting ability. They are very hard to kill and the most challenging enemy ships. 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 enemy ships. Similarly, if a Tier 3 – Evolved Version enemy is destroyed, it splits into Tier 1 – Evolved Version and Tier 2-Evolved version enemy ships.

2.6.2 Enemy Space Stations

Enemy Space Stations are the places where enemy spaceships spawn at a certain frequency. There are two categories of enemy spaceships. 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 space stations where Evolved Versions are created in order to make themselves stronger and “evolved”.

2.6.3 Commander of the Levels

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

2.7 Alliance Space Stations

Alliance Space Stations are significant war bases where allies work. If an alliance station is destroyed by enemies, allies in the station die and then, the player loses point. In order to get high score, the player should prevent enemies from damaging alliance stations.

2.8 Settings

In this menu, the player can adjust sound volume. Also, keyboard settings can be changed. In this way, for player actions, desirable keys can be assigned. Lastly, to be more user-friendly, there is color-blind option to change the game color.

3. Functional Requirements

3.1 Play Game

The game will have one player and two player mode. 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 spaceship and the level s/he wants to play.

At the beginning, all levels will be locked except first level. After passing a level, 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 spaceship can move only in right and left direction. However, the player has ability to arrange altitude. Spaceship can fire in the direction of movement. The player tries to destroy all enemy space stations using these abilities. The player will be able to find the places of enemy space stations by looking on the radar screen. While doing this, there will be enemies spreading from their space stations and trying to eliminate player’s spaceship. The player has to shoot these enemies and escape from getting hit by them. If the player gets hit, s/he loses some of health points. After destroying all enemy space stations, the player has to destroy the commander of this level. By the way, the player also has to protect alliance space stations from enemies continuously attacking them. For each alliances space station destroyed by enemies, the player loses point. If all alliances space stations are destroyed before the level is completed or the spaceship loses all health points, the game ends. To succeed in the game, the player can use bonuses show up after eliminating enemies.

3.1.2. Multiplayer Game

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

Two players will fight against enemies together. Their health points will be independent from each other. If one of the spaceships loses all health points, the other one will be able to continue to the game. Their scores will be calculated separately.

3.2 Bonuses

There will be various bonuses which show up after destroying enemies. These bonuses and their qualities explained in 2.5. The player will be able to gain these bonuses by just passing over them after they appear. Using bonuses, the game will be more competitive and excited. After taking bonuses, the player has to immediately use them except for smart bomb and hyperjump bonuses. They can be used whenever the player wants.

3.3 How to Play

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

- Rules
- Different Enemy Types
- Commanders of Each Level
- Keyboard Controls
- Bonuses

3.4 Options

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

- Adjusting the volume
- Turning on or off the volume
- Key settings

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

3.5 Pause

After the game starts, the player can pause the game by pressing pause button in the map. 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.

3.6 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.

3.7 High Scores

The player can access ten of the highest points people win through the game. This quality will make people more competitive to have a place in the list.

3.8 Score Calculation

When the player destroys enemy space stations, s/he gains point. The amount of points the player wins decrease as time passes. The player should try to eliminate all enemy space stations quickly to get a high score. Also, people are killed when alliance space stations are hit by enemies. The player loses points when people die. If one of alliance space stations are eliminated, the player loses more point.

3.9 Extendibility

3.9.1 Levels

Our game will currently have three levels. However, we can add more challenging levels to the game.

3.9.2 Enemy Types

Our game will have 3 types of enemies. We can add more enemy types with different characteristics to make the game more interesting and challenging.

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 30-60 fps according to the system that runs the game. Especially, with the systems that have better graphic cards, the game will have better performance. We are planning to design the animations 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]. With this way, the variety and number of our user base as well as portability of our game will increase. Furthermore, different bugs for different operating systems will be less than average applications.

4.3 Extendibility

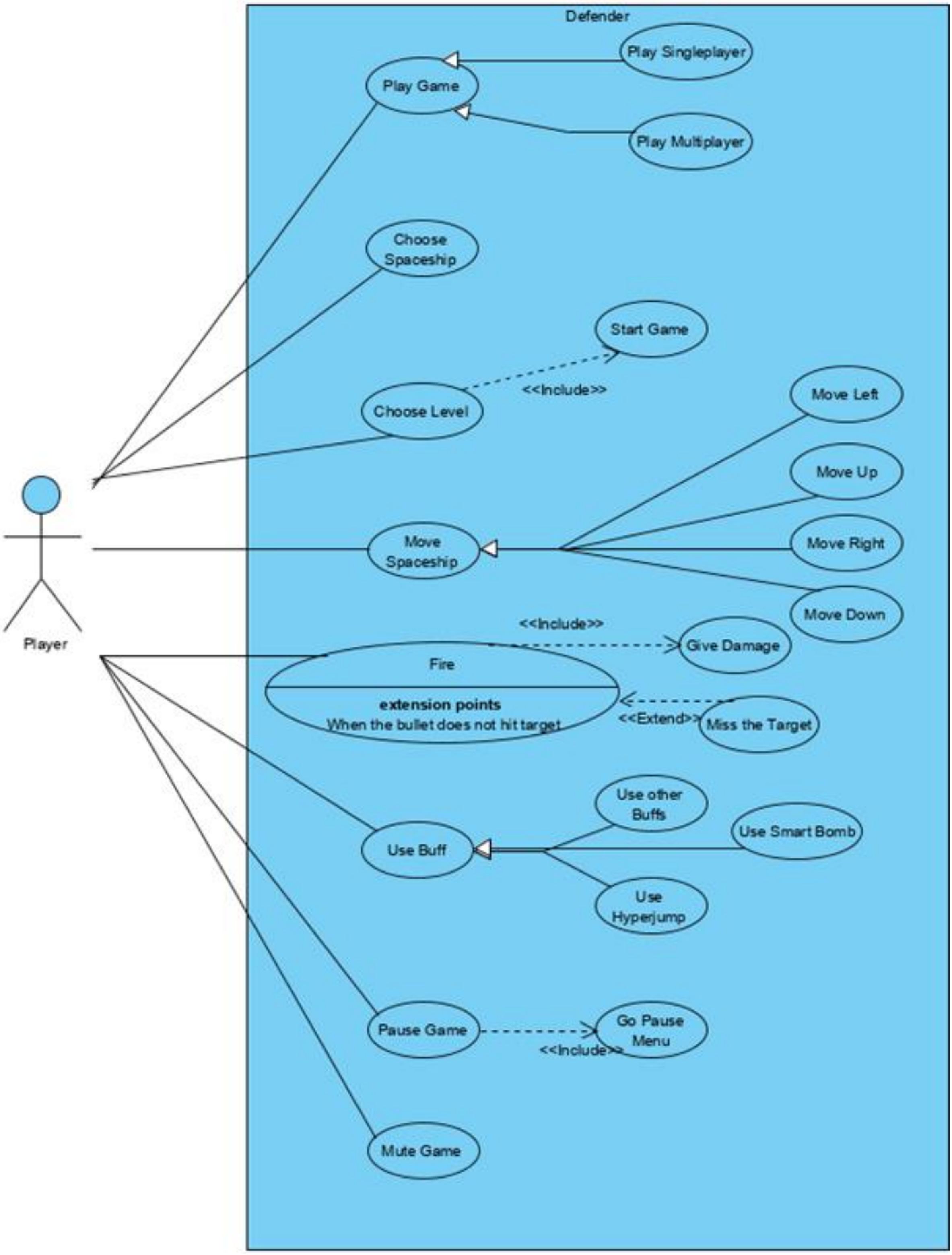
Our game is designed suitable to add new features or extend existing features easily without having major bugs or obstacles for next versions. In order to achieve this goal, we designed our object class diagrams compatible for prospect new codes as it can be seen in section 5.1. For example, if there 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 User Friendly Interface

We are aware of the importance of users for our game to be successful. Therefore, one of our major concerns is building a user-friendly interface. In order to achieve this goal, we have two approaches which are designing customizable and plain user interface. For example, for users who are color-blind there will be options of color themes. Of course, normal users can also use different themes. Similarly, there will be options for key bindings or audio. For users who are new to game there will be information page which explains how to play the game and what needs to be done to finish levels.

5. System Models

5.1 Use Case Models


Defender
Play Game
Play Singleplayer
Play Multiplayer
Choose Spaceship
Start Game
<<Include>>
Choose Level
Move Left
Move Up
Move Right
Move Down
Move Spaceship
Player
<<Include>>
Give Damage
Fire
extension points
When the bullet does not hit target
<<Extend>>
Miss the Target
Use other Buffs
Use Smart Bomb
Use Buff
Use Hyperjump
Pause Game
Go Pause Menu
<<Include>>
Mute 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 multiplayer options to play game**. According chosen option, the game provides different choices for user to **choose his/her spaceship**. 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 spaceship and level is determined, the player can start to play game. In the game user can **move his/her spaceship** to **up, down, right, and left** directions. Also, **the player can fire** with his/her spaceship against enemies' spaceships. The player also can **use buffs** to improve the spaceships 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 Singleplayer or Multiplayer options

Post-conditions:

- Player is successfully directed to spaceship selection part.

Main Event Flow:

- Player who is in the main menu clicks the Singleplayer or Multiplayer buttons
- System responses and displays spaceship selection menu for user

Alternative Event Flow(**s**):

- Other Menu Options

Use Case #2

Use Case: Choose Spaceship

Participating Actor: Player

Pre-conditions:

- Player should have decided that s/he plays the game whether multiplayer or single player

Post-conditions:

- Spaceships that will be shown in the game is successfully determined

Main Event Flow:

- Player chooses a spaceship among given options
- System stores information of the chosen spaceship 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 spaceship

Use Case #3

Use Case: Choose Episode

Participating Actor: Player

Pre-conditions:

- Player must have chosen spaceship 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 spaceship.

Alternative Event Flow(**s**):

- Player can back to spaceship selection screen

Use Case #4

Use Case: Move Spaceship

Participating Actor: Player

Pre-conditions:

- The game must have started
- Player must press the keys which move spaceship

Post-conditions:

- Spaceship moves successfully according to pressed key

Main Event Flow:

- Player presses the keys in order to move spaceship
- System changes the coordinates of spaceship 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 spaceship improved by used buff

Main Event Flow:

- In order to get buff, Player must kill enemy spaceships
- System drops buff randomly from enemy spaceships
- Player collects the buff that is dropped from enemy ship
- System puts collected buff to improvement slot
- Player uses this buff by pushing proper button
- System improves spaceship according to used 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

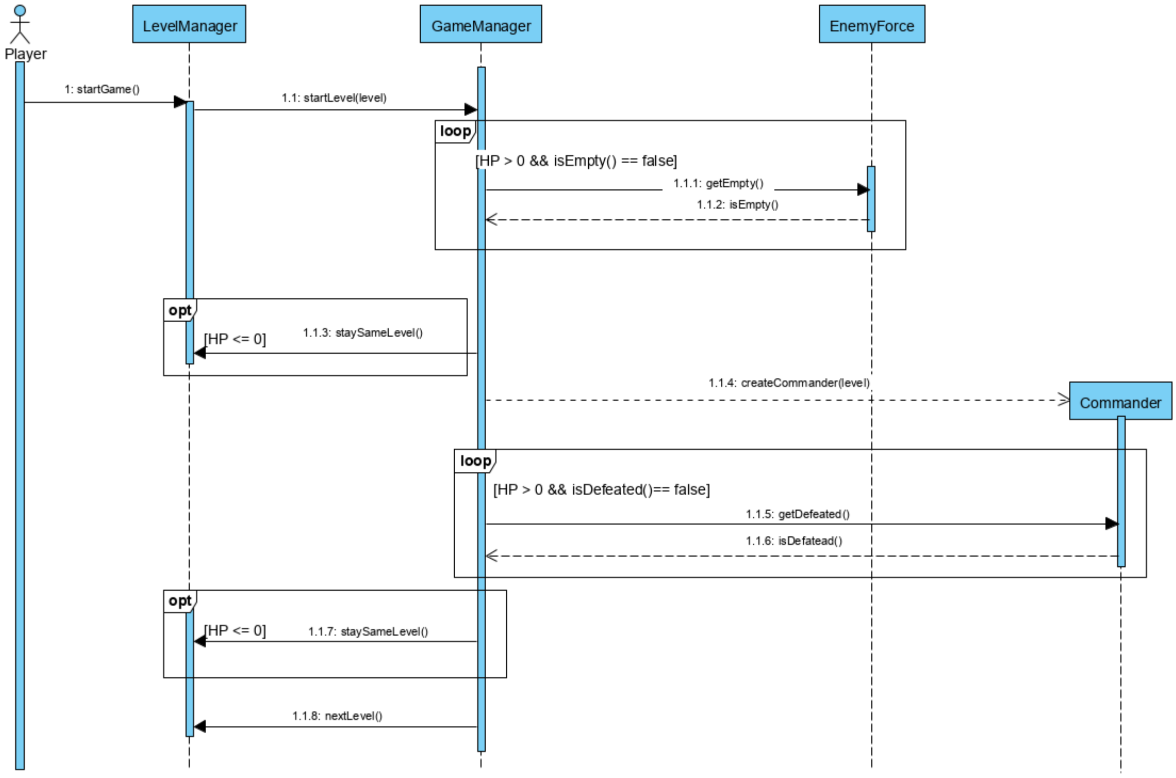
5.2 Dynamic Models

5.2.1 Sequence Diagrams

5.2.1.1 Passing a Level

Scenario: The mechanism of passing a level

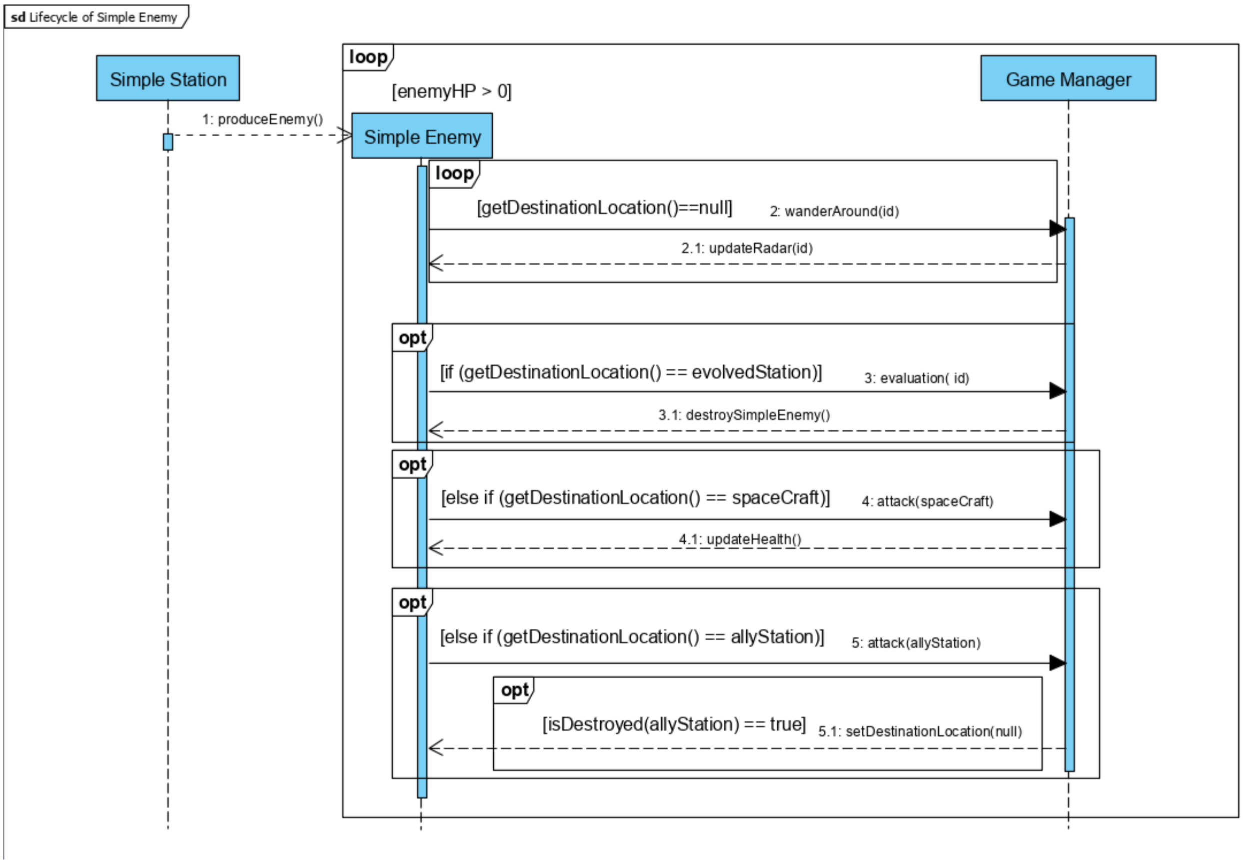
The player opens the game and starts a level. The player plays the game until he/she dies or kills the entire enemy forces successfully. If the player dies, level is not completed, and he/she has to play same level again. If the player successfully destroys all enemies, commander of the level spawns. Now the player has to defeat the commander without being dead. If commander is defeated, the player can start new level.



5.2.1.2 Lifecycle of Single Enemy

Scenario: How a single enemy behaves in the game.

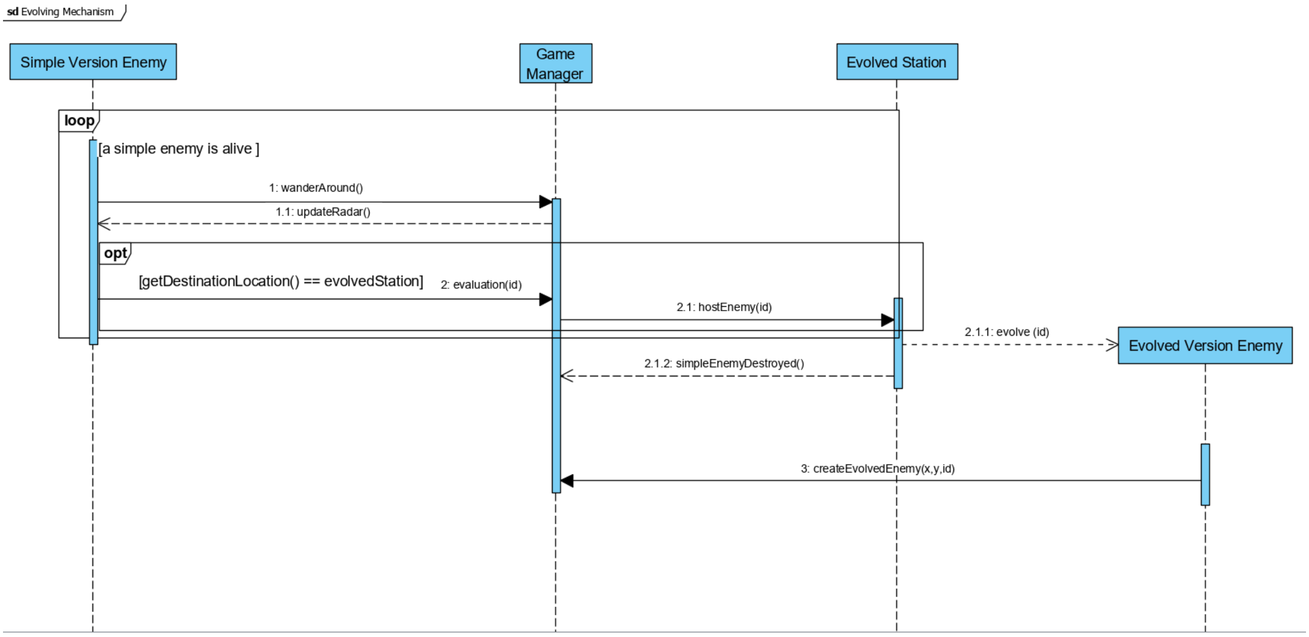
A simple enemy is created in the simple station. It starts to wander around the map. While it is moving, its radar always being updated according to environment. When it finds a meaningful target, it changes its behavior. If it finds an evolved station, it goes to the evolved station to evolve. If it finds the player, it starts attacking until it kills the player, or the player gets off its radar. Lastly, if it finds allied station – it is allied station in player’s perspective- it starts attacking it. A simple enemy performs all of the activities listed above as long as it is not killed.


sd Lifecycle of Simple Enemy
Simple Station
loop
[enemyHP > 0]
Game Manager
1: produceEnemy()
Simple Enemy
loop
[getDestinationLocation()==null]
2: wanderAround(id)
2.1: updateRadar(id)
opt
[if (getDestinationLocation() == evolvedStation)]
3: evaluation(id)
3.1: destroySimpleEnemy()
opt
[else if (getDestinationLocation() == spaceCraft)]
4: attack(spaceCraft)
4.1: updateHealth()
opt
[else if (getDestinationLocation() == allyStation)]
5: attack(allyStation)
opt
[isDestroyed(allyStation) == true]
5.1: setDestinationLocation(null)

5.2.1.3 Evolving Mechanism

Scenario: What happens in the evaluation process.

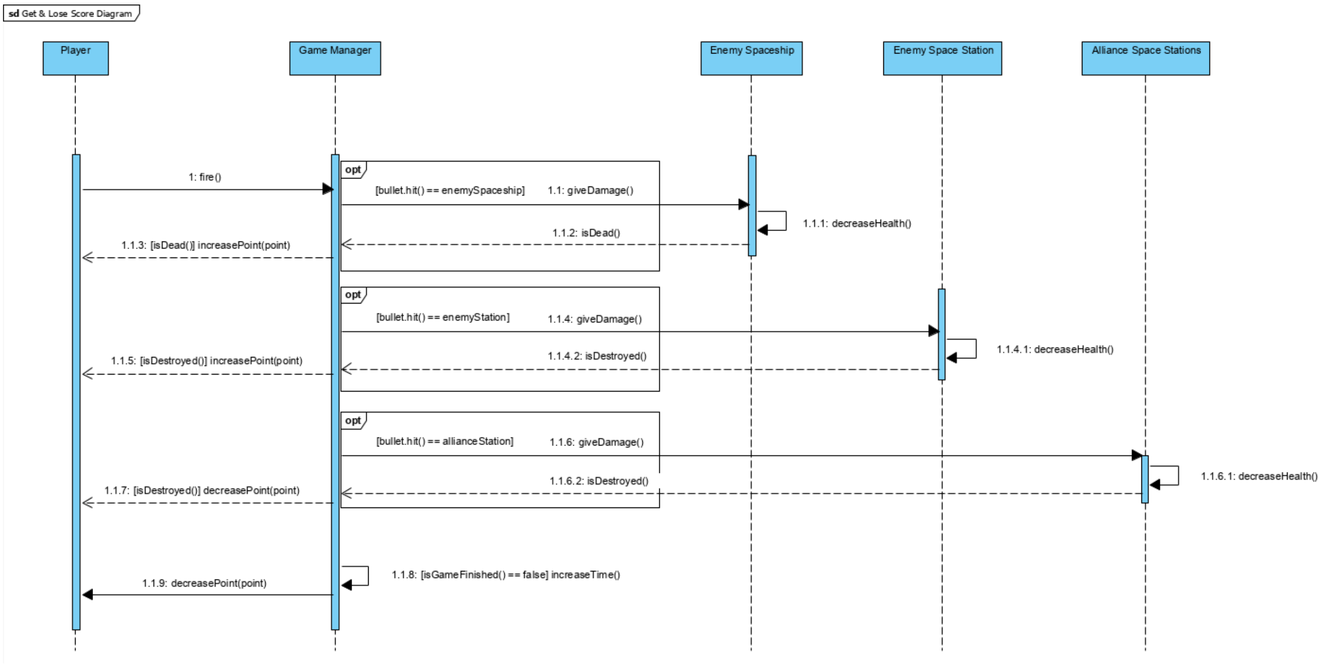
While a simple enemy is wandering it can find an evolved station. When an evolved station is found, the simple enemy changes its target as evolved station. The evolved station hosts the enemy and improves its attributes. As a result of the process, an evolved enemy is released to the map.



5.2.1.4 Get & Lose Point Mechanism

Scenario: How the player can get and lose points

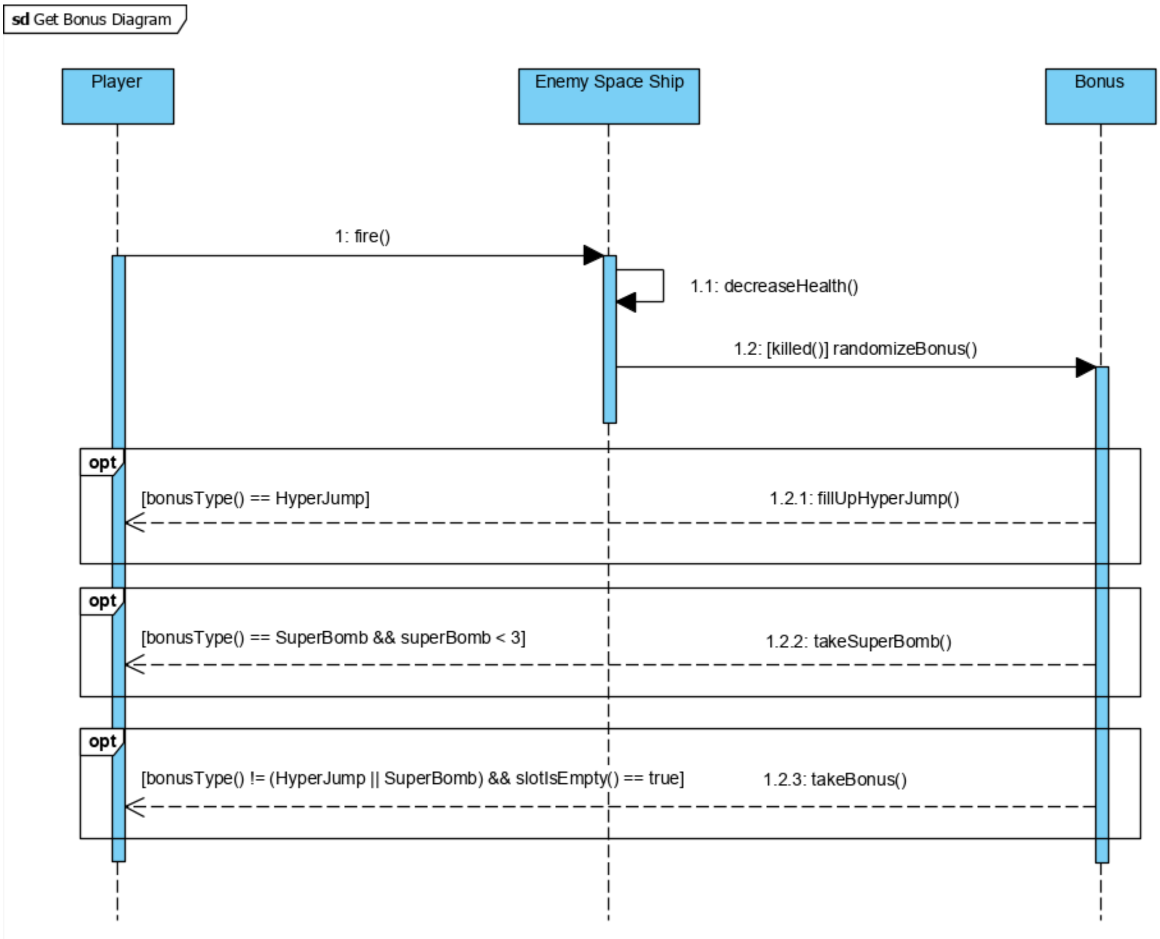
When the player fires, game manager checks whether the bullet hits something or not. If it doesn't hit anything, game manager will not do anything. If it hits an enemy spaceship, it gives damage to the spaceship. Then, game manager checks whether this spaceship is destroyed. If your fire can destroy the enemy spaceship, game manager increases your total point. If the bullet hits an enemy space station, the same process will take place. However, if the bullet hits an alliance station, the station will take damage. If the alliance station is destroyed somehow, the player loses point since the player should try to protect alliance forces. It doesn't matter who destroys alliance stations. Lastly, during the game, game manager increases time elapsed and every time passing causes a decrease in total point.


sd Get & Lose Score Diagram
Player
Game Manager
Enemy Spaceship
Enemy Space Station
Alliance Space Stations
opt
1: fire()
[bullet.hit() == enemySpaceship]
1.1: giveDamage()
1.1.1: decreaseHealth()
1.1.2: isDead()
1.1.3: [isDead()] increasePoint(point)
opt
[bullet.hit() == enemyStation]
1.1.4: giveDamage()
1.1.4.1: decreaseHealth()
1.1.4.2: isDestroyed()
1.1.5: [isDestroyed()] increasePoint(point)
opt
[bullet.hit() == allianceStation]
1.1.6: giveDamage()
1.1.6.1: decreaseHealth()
1.1.6.2: isDestroyed()
1.1.7: [isDestroyed()] decreasePoint(point)
1.1.8: [isGameFinished() == false] increaseTime()
1.1.9: decreasePoint(point)

5.2.1.5 Take Bonus Mechanism

Scenario: How the player can take bonus

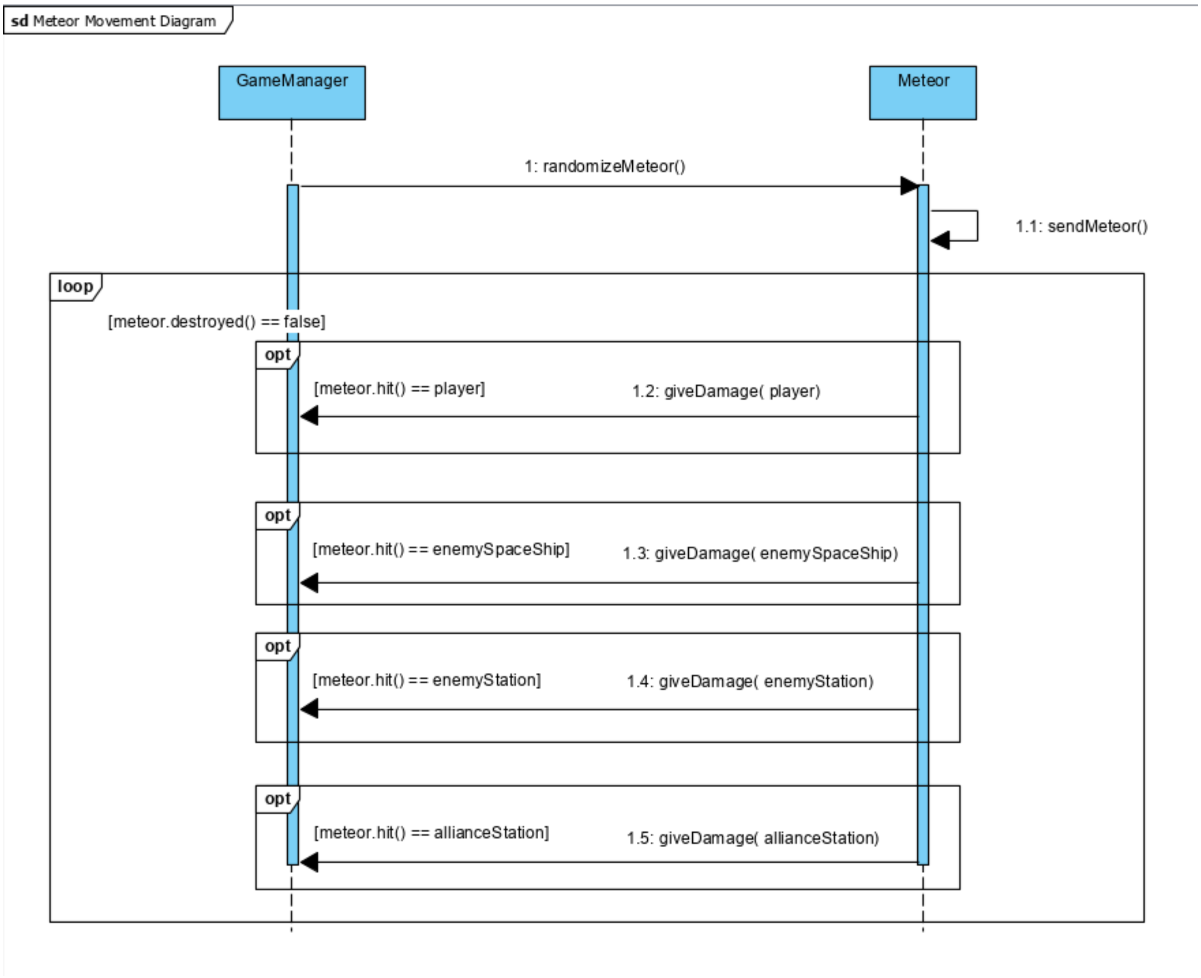
When the player fires to an enemy spaceship, the enemy ship's HP will decrease. If the player kills the enemy ship, a bonus can be generated as a prize. If this bonus is hyper jump ability, the player's rechargeable hyper jump ability will be automatically filled up. If this is superbomb bonus and the number of the player's current bomb is less than 3, one extra superbomb will be taken to use later. Lastly, if bonus type is not one of these two options, the player checks the slot since he/she can have only one ability in the slot at the same time. So, the player can take the bonus only if the slot is empty.


sd Get Bonus Diagram
Player
Enemy Space Ship
Bonus
1: fire()
1.1: decreaseHealth()
1.2: [killed()] randomizeBonus()
opt
[bonusType() == HyperJump]
1.2.1: fillUpHyperJump()
opt
[bonusType() == SuperBomb && superBomb < 3]
1.2.2: takeSuperBomb()
opt
[bonusType() != (HyperJump || SuperBomb) && slotIsEmpty() == true]
1.2.3: takeBonus()

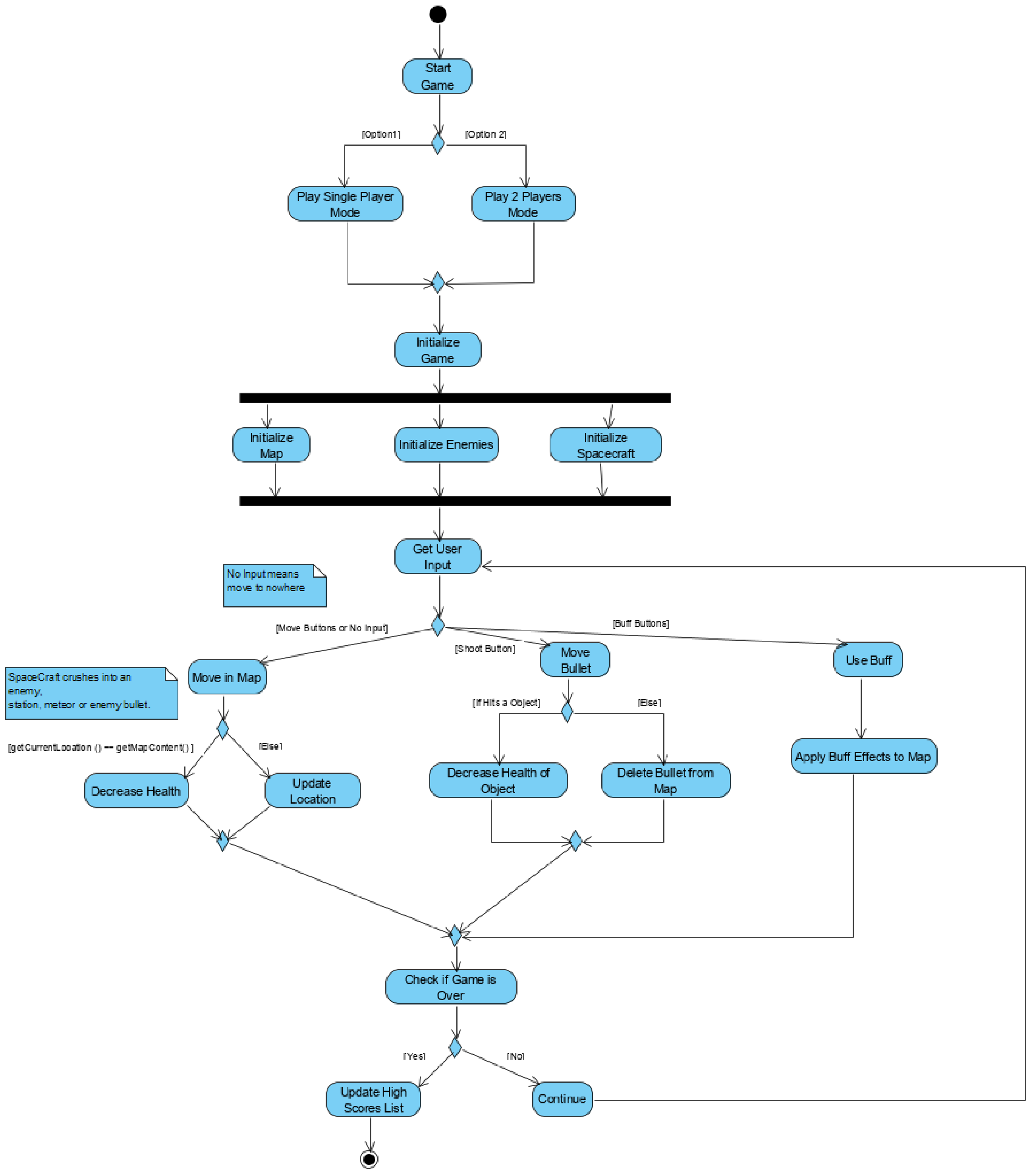
5.2.1.6 Meteor Fall Mechanism

Scenario: How meteor behaves

Sometimes during the game, game manager can create meteor. If a meteor is created, game manager checks whether it hits something until it leaves the map and is destroyed. The meteor can hit the player, enemy spaceships, enemy stations, or alliance stations. When it hits one of them, the meteor gives damage what it hits.


sd Meteor Movement Diagram
GameManager
Meteor
1: randomizeMeteor()
1.1: sendMeteor()
loop
[meteor.destroyed() == false]
opt
[meteor.hit() == player]
1.2: giveDamage(player)
opt
[meteor.hit() == enemySpaceShip]
1.3: giveDamage(enemySpaceShip)
opt
[meteor.hit() == enemyStation]
1.4: giveDamage(enemyStation)
opt
[meteor.hit() == allianceStation]
1.5: giveDamage(allianceStation)

5.2.2 Activity Diagram



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

At the beginning, the player starts the game. There are two modes of the game, single player mode and 2 players mode. According to the choice of the player corresponding mode starts. No matter which mode is picked, game is initialized. Then, objects that are required to play game are initialized in parallel such as Map, Enemies and Spacecraft objects.

Now the game is started, and Game Manager waits for an input from the player. There are mainly three options the player has:

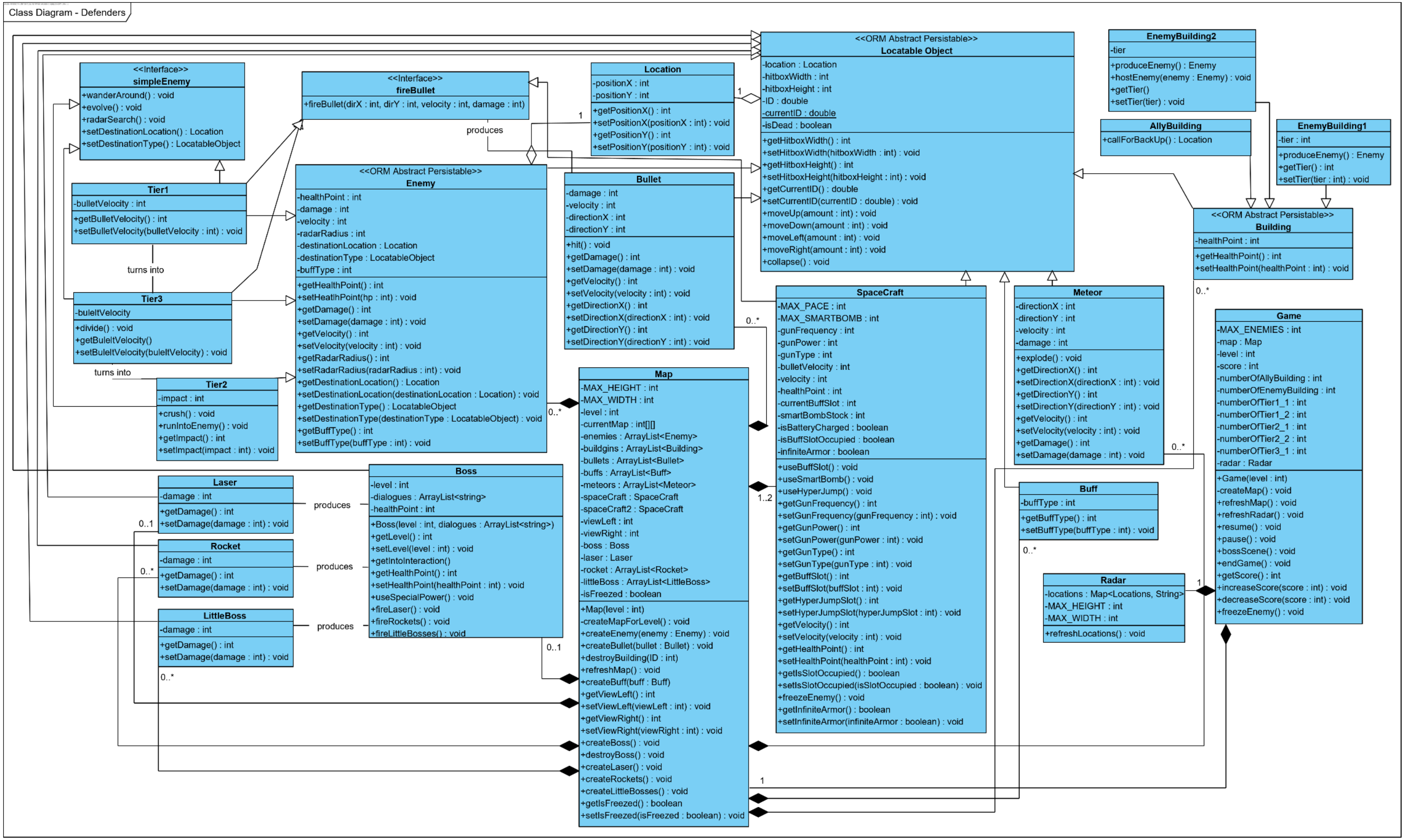
Press Move Keys or Do Nothing: Firstly, do nothing means stay at the same location. In other words, it is the base case of moving somewhere. Therefore, it can be treated as moving. If the current location of Spacecraft is same as any other object -except itself and buffs-, it crashes the object. The object can be enemy spaceship, enemy station, ally station, meteor or enemy bullet. In this case decrease health of the Spacecraft. If Spacecraft does not crash anything, it is location is updated.

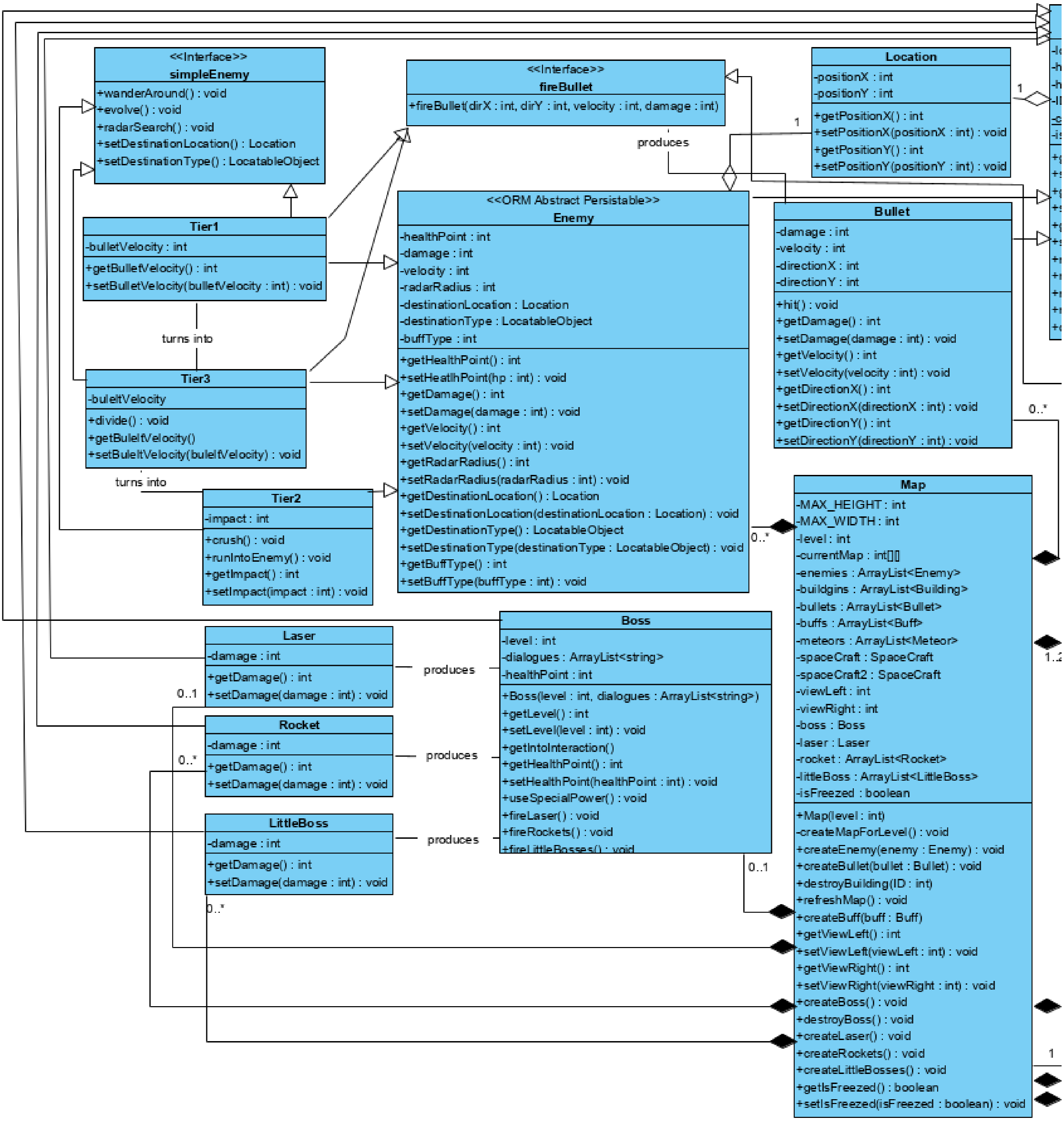
Press Shoot Key: If the bullet hits something -again it can be enemy, a station etc. -, object's health is decreased. If the bullet does not hit anything, it leaves the map. After it leaves, the bullet object is destroyed.

Press Buff Buttons: There are some buffs that the player can use and each has a different effect in the game. When the player hits one of the buff keys, corresponding buff is applied to the game.

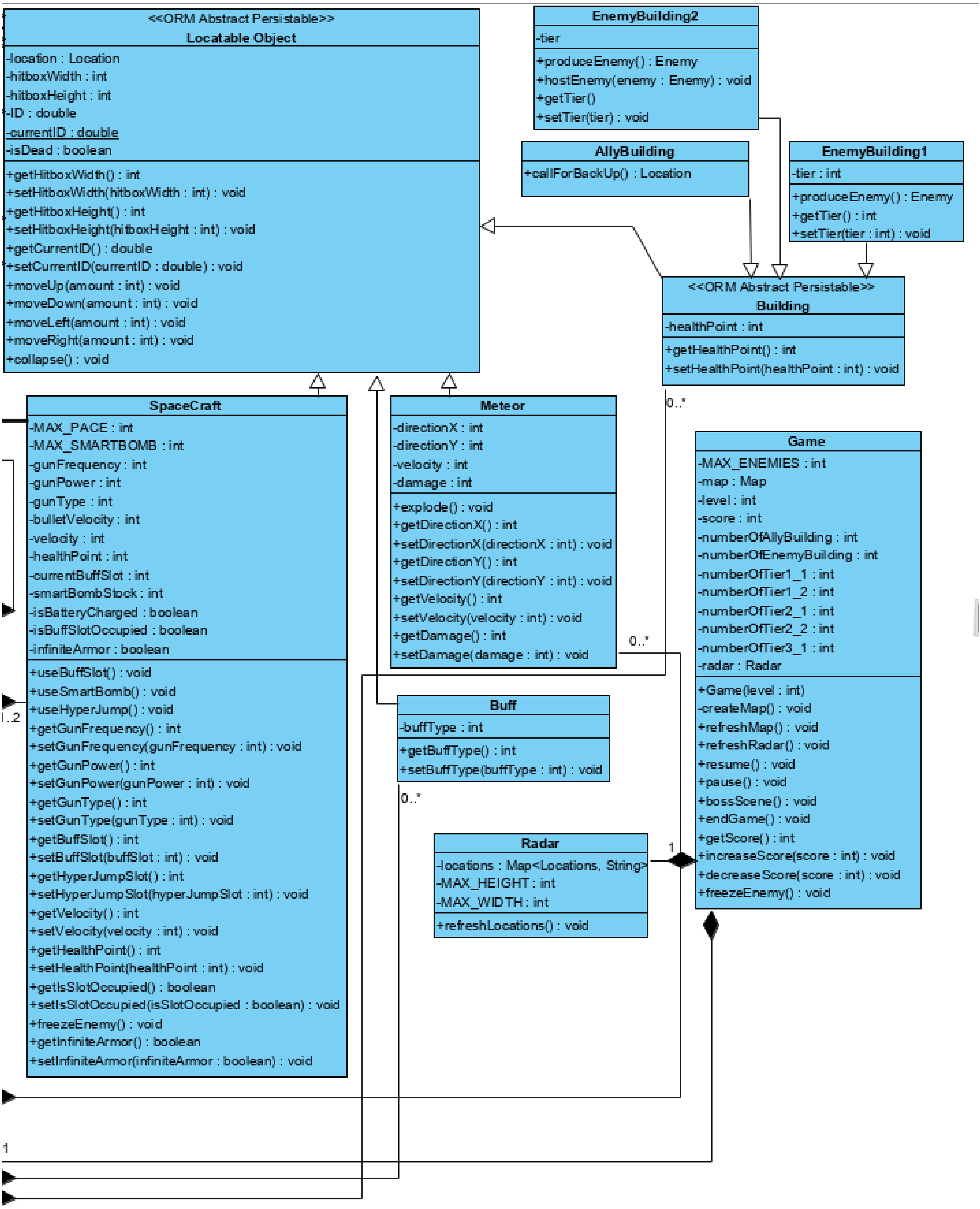
After acting according to the user input, it is checked whether the game is over or not. If the game is over, scoreboard is updated, and game is end. If it is not over, the game is continuing, and the Game Manager waits for the next input.

5.3 Object and Class Model


Class Diagram - Defenders
<<Interface>>
simpleEnemy
+wanderAround() : void
+evolve() : void
+radarSearch() : void
+setDestinationLocation() : Location
+setDestinationType() : LocatableObject
<<Interface>>
fireBullet
+fireBullet(dirX : int, dirY : int, velocity : int, damage : int)
produces
Location
-positionX : int
-positionY : int
+getPositionX() : int
+setPositionX(positionX : int) : void
+getPositionY() : int
+setPositionY(positionY : int) : void
1
1
<<ORM Abstract Persistable>>
Locatable Object
-location : Location
-hitboxWidth : int
-hitboxHeight : int
-ID : double
-currentID : double
-isDead : boolean
+getHitboxWidth() : int
+setHitboxWidth(hitboxWidth : int) : void
+getHitboxHeight() : int
+setHitboxHeight(hitboxHeight : int) : void
+getCurrentID() : double
+setCurrentID(currentID : double) : void
+moveUp(amount : int) : void
+moveDown(amount : int) : void
+moveLeft(amount : int) : void
+moveRight(amount : int) : void
+collapse() : void
EnemyBuilding2
-tier
+produceEnemy() : Enemy
+hostEnemy(enemy : Enemy) : void
+getTier()
+setTier(tier) : void
AllyBuilding
+callForBackUp() : Location
EnemyBuilding1
-tier : int
+produceEnemy() : Enemy
+getTier() : int
+setTier(tier : int) : void
<<ORM Abstract Persistable>>
Building
-healthPoint : int
+getHealthPoint() : int
+setHealthPoint(healthPoint : int) : void
0..*
Tier1
-bulletVelocity : int
+getBulletVelocity() : int
+setBulletVelocity(bulletVelocity : int) : void
turns into
Tier3
-buleltVelocity
+divide() : void
+getBuleltVelocity()
+setBuleltVelocity(buleltVelocity) : void
turns into
Tier2
-impact : int
+crush() : void
+runIntoEnemy() : void
+getImpact() : int
+setImpact(impact : int) : void
<<ORM Abstract Persistable>>
Enemy
-healthPoint : int
-damage : int
-velocity : int
-radarRadius : int
-destinationLocation : Location
-destinationType : LocatableObject
-buffType : int
+getHealthPoint() : int
+setHeathPoint(hp : int) : void
+getDamage() : int
+setDamage(damage : int) : void
+getVelocity() : int
+setVelocity(velocity : int) : void
+getRadarRadius() : int
+setRadarRadius(radarRadius : int) : void
+getDestinationLocation() : Location
+setDestinationLocation(destinationLocation : Location) : void
+getDestinationType() : LocatableObject
+setDestinationType(destinationType : LocatableObject) : void
+getBuffType() : int
+setBuffType(buffType : int) : void
0..*
Bullet
-damage : int
-velocity : int
-directionX : int
-directionY : int
+hit() : void
+getDamage() : int
+setDamage(damage : int) : void
+getVelocity() : int
+setVelocity(velocity : int) : void
+getDirectionX() : int
+setDirectionX(directionX : int) : void
+getDirectionY() : int
+setDirectionY(directionY : int) : void
0..*
SpaceCraft
-MAX_PACE : int
-MAX_SMARTBOMB : int
-gunFrequency : int
-gunPower : int
-gunType : int
-bulletVelocity : int
-velocity : int
-healthPoint : int
-currentBuffSlot : int
-smartBombStock : int
-isBatteryCharged : boolean
-isBuffSlotOccupied : boolean
-infiniteArmor : boolean
+useBuffSlot() : void
+useSmartBomb() : void
+useHyperJump() : void
+getGunFrequency() : int
+setGunFrequency(gunFrequency : int) : void
+getGunPower() : int
+setGunPower(gunPower : int) : void
+getGunType() : int
+setGunType(gunType : int) : void
+getBuffSlot() : int
+setBuffSlot(buffSlot : int) : void
+getHyperJumpSlot() : int
+setHyperJumpSlot(hyperJumpSlot : int) : void
+getVelocity() : int
+setVelocity(velocity : int) : void
+getHealthPoint() : int
+setHealthPoint(healthPoint : int) : void
+getIsSlotOccupied() : boolean
+setIsSlotOccupied(isSlotOccupied : boolean) : void
+freezeEnemy() : void
+getInfiniteArmor() : boolean
+setInfiniteArmor(infiniteArmor : boolean) : void
1..2
Meteor
-directionX : int
-directionY : int
-velocity : int
-damage : int
+explode() : void
+getDirectionX() : int
+setDirectionX(directionX : int) : void
+getDirectionY() : int
+setDirectionY(directionY : int) : void
+getVelocity() : int
+setVelocity(velocity : int) : void
+getDamage() : int
+setDamage(damage : int) : void
0..*
Buff
-buffType : int
+getBuffType() : int
+setBuffType(buffType : int) : void
0..*
Radar
-locations : Map<Locations, String>
-MAX_HEIGHT : int
-MAX_WIDTH : int
+refreshLocations() : void
1
Game
-MAX_ENEMIES : int
-map : Map
-level : int
-score : int
-numberOfAllyBuilding : int
-numberOfEnemyBuilding : int
-numberOfTier1_1 : int
-numberOfTier1_2 : int
-numberOfTier2_1 : int
-numberOfTier2_2 : int
-numberOfTier3_1 : int
-radar : Radar
+Game(level : int)
-createMap() : void
+refreshMap() : void
+refreshRadar() : void
+resume() : void
+pause() : void
+bossScene() : void
+endGame() : void
+getScore() : int
+increaseScore(score : int) : void
+decreaseScore(score : int) : void
+freezeEnemy() : void
1
Map
-MAX_HEIGHT : int
-MAX_WIDTH : int
-level : int
-currentMap : int[][]
-enemies : ArrayList<Enemy>
-buildgins : ArrayList<Building>
-bullets : ArrayList<Bullet>
-buffs : ArrayList<Buff>
-meteors : ArrayList<Meteor>
-spaceCraft : SpaceCraft
-spaceCraft2 : SpaceCraft
-viewLeft : int
-viewRight : int
-boss : Boss
-laser : Laser
-rocket : ArrayList<Rocket>
-littleBoss : ArrayList<LittleBoss>
-isFreezed : boolean
+Map(level : int)
-createMapForLevel() : void
+createEnemy(enemy : Enemy) : void
+createBullet(bullet : Bullet) : void
+destroyBuilding(ID : int)
+refreshMap() : void
+createBuff(buff : Buff)
+getViewLeft() : int
+setViewLeft(viewLeft : int) : void
+getViewRight() : int
+setViewRight(viewRight : int) : void
+createBoss() : void
+destroyBoss() : void
+createLaser() : void
+createRockets() : void
+createLittleBosses() : void
+getIsFreezed() : boolean
+setIsFreezed(isFreezed : boolean) : void
Boss
-level : int
-dialogues : ArrayList<string>
-healthPoint : int
+Boss(level : int, dialogues : ArrayList<string>)
+getLevel() : int
+setLevel(level : int) : void
+getIntoInteraction()
+getHealthPoint() : int
+setHealthPoint(healthPoint : int) : void
+useSpecialPower() : void
+fireLaser() : void
+fireRockets() : void
+fireLittleBosses() : void
0..1
Laser
-damage : int
+getDamage() : int
+setDamage(damage : int) : void
produces
0..1
Rocket
-damage : int
+getDamage() : int
+setDamage(damage : int) : void
produces
0..*
LittleBoss
-damage : int
+getDamage() : int
+setDamage(damage : int) : void
produces
0..*



<<Interface>>
simpleEnemy
+wanderAround() : void
+evolve() : void
+radarSearch() : void
+setDestinationLocation() : Location
+setDestinationType() : LocatableObject
<<Interface>>
fireBullet
+fireBullet(dirX : int, dirY : int, velocity : int, damage : int)
Location
-positionX : int
-positionY : int
+getPositionX() : int
+setPositionX(positionX : int) : void
+getPositionY() : int
+setPositionY(positionY : int) : void
1
1
produces
Tier1
-bulletVelocity : int
+getBulletVelocity() : int
+setBulletVelocity(bulletVelocity : int) : void
turns into
Tier3
-buleltVelocity
+divide() : void
+getBuleltVelocity()
+setBuleltVelocity(buleltVelocity) : void
turns into
Tier2
-impact : int
+crush() : void
+runIntoEnemy() : void
+getImpact() : int
+setImpact(impact : int) : void
<<ORM Abstract Persistable>>
Enemy
-healthPoint : int
-damage : int
-velocity : int
-radarRadius : int
-destinationLocation : Location
-destinationType : LocatableObject
-buffType : int
+getHealthPoint() : int
+setHeatlhPoint(hp : int) : void
+getDamage() : int
+setDamage(damage : int) : void
+getVelocity() : int
+setVelocity(velocity : int) : void
+getRadarRadius() : int
+setRadarRadius(radarRadius : int) : void
+getDestinationLocation() : Location
+setDestinationLocation(destinationLocation : Location) : void
+getDestinationType() : LocatableObject
+setDestinationType(destinationType : LocatableObject) : void
+getBuffType() : int
+setBuffType(buffType : int) : void
Bullet
-damage : int
-velocity : int
-directionX : int
-directionY : int
+hit() : void
+getDamage() : int
+setDamage(damage : int) : void
+getVelocity() : int
+setVelocity(velocity : int) : void
+getDirectionX() : int
+setDirectionX(directionX : int) : void
+getDirectionY() : int
+setDirectionY(directionY : int) : void
0..*
0..*
Map
-MAX_HEIGHT : int
-MAX_WIDTH : int
-level : int
-currentMap : int[][]
-enemies : ArrayList<Enemy>
-buildgins : ArrayList<Building>
-bullets : ArrayList<Bullet>
-buffs : ArrayList<Buff>
-meteors : ArrayList<Meteor>
-spaceCraft : SpaceCraft
-spaceCraft2 : SpaceCraft
-viewLeft : int
-viewRight : int
-boss : Boss
-laser : Laser
-rocket : ArrayList<Rocket>
-littleBoss : ArrayList<LittleBoss>
-isFreezed : boolean
+Map(level : int)
-createMapForLevel() : void
+createEnemy(enemy : Enemy) : void
+createBullet(bullet : Bullet) : void
+destroyBuilding(ID : int)
+refreshMap() : void
+createBuff(buff : Buff)
+getViewLeft() : int
+setViewLeft(viewLeft : int) : void
+getViewRight() : int
+setViewRight(viewRight : int) : void
+createBoss() : void
+destroyBoss() : void
+createLaser() : void
+createRockets() : void
+createLittleBosses() : void
+getIsFreezed() : boolean
+setIsFreezed(isFreezed : boolean) : void
1..
1
Boss
-level : int
-dialogues : ArrayList<string>
-healthPoint : int
+Boss(level : int, dialogues : ArrayList<string>)
+getLevel() : int
+setLevel(level : int) : void
+getIntoInteraction()
+getHealthPoint() : int
+setHealthPoint(healthPoint : int) : void
+useSpecialPower() : void
+fireLaser() : void
+fireRockets() : void
+fireLittleBosses() : void
0..1
Laser
-damage : int
+getDamage() : int
+setDamage(damage : int) : void
produces
0..1
Rocket
-damage : int
+getDamage() : int
+setDamage(damage : int) : void
produces
0..*
LittleBoss
-damage : int
+getDamage() : int
+setDamage(damage : int) : void
produces
0..*



<<ORM Abstract Persistable>>
Locatable Object
-location : Location
-hitboxWidth : int
-hitboxHeight : int
-ID : double
-currentID : double
-isDead : boolean
+getHitboxWidth() : int
+setHitboxWidth(hitboxWidth : int) : void
+getHitboxHeight() : int
+setHitboxHeight(hitboxHeight : int) : void
+getCurrentID() : double
+setCurrentID(currentID : double) : void
+moveUp(amount : int) : void
+moveDown(amount : int) : void
+moveLeft(amount : int) : void
+moveRight(amount : int) : void
+collapse() : void
EnemyBuilding2
-tier
+produceEnemy() : Enemy
+hostEnemy(enemy : Enemy) : void
+getTier()
+setTier(tier) : void
AllyBuilding
+callForBackUp() : Location
EnemyBuilding1
-tier : int
+produceEnemy() : Enemy
+getTier() : int
+setTier(tier : int) : void
<<ORM Abstract Persistable>>
Building
-healthPoint : int
+getHealthPoint() : int
+setHealthPoint(healthPoint : int) : void
0..*
SpaceCraft
-MAX_PACE : int
-MAX_SMARTBOMB : int
-gunFrequency : int
-gunPower : int
-gunType : int
-bulletVelocity : int
-velocity : int
-healthPoint : int
-currentBuffSlot : int
-smartBombStock : int
-isBatteryCharged : boolean
-isBuffSlotOccupied : boolean
-infiniteArmor : boolean
+useBuffSlot() : void
+useSmartBomb() : void
+useHyperJump() : void
+getGunFrequency() : int
+setGunFrequency(gunFrequency : int) : void
+getGunPower() : int
+setGunPower(gunPower : int) : void
+getGunType() : int
+setGunType(gunType : int) : void
+getBuffSlot() : int
+setBuffSlot(buffSlot : int) : void
+getHyperJumpSlot() : int
+setHyperJumpSlot(hyperJumpSlot : int) : void
+getVelocity() : int
+setVelocity(velocity : int) : void
+getHealthPoint() : int
+setHealthPoint(healthPoint : int) : void
+getIsSlotOccupied() : boolean
+setIsSlotOccupied(isSlotOccupied : boolean) : void
+freezeEnemy() : void
+getInfiniteArmor() : boolean
+setInfiniteArmor(infiniteArmor : boolean) : void
1..2
Meteor
-directionX : int
-directionY : int
-velocity : int
-damage : int
+explode() : void
+getDirectionX() : int
+setDirectionX(directionX : int) : void
+getDirectionY() : int
+setDirectionY(directionY : int) : void
+getVelocity() : int
+setVelocity(velocity : int) : void
+getDamage() : int
+setDamage(damage : int) : void
0..*
Buff
-buffType : int
+getBuffType() : int
+setBuffType(buffType : int) : void
0..*
Radar
-locations : Map<Locations, String>
-MAX_HEIGHT : int
-MAX_WIDTH : int
+refreshLocations() : void
1
Game
-MAX_ENEMIES : int
-map : Map
-level : int
-score : int
-numberOfAllyBuilding : int
-numberOfEnemyBuilding : int
-numberOfTier1_1 : int
-numberOfTier1_2 : int
-numberOfTier2_1 : int
-numberOfTier2_2 : int
-numberOfTier3_1 : int
-radar : Radar
+Game(level : int)
-createMap() : void
+refreshMap() : void
+refreshRadar() : void
+resume() : void
+pause() : void
+bossScene() : void
+endGame() : void
+getScore() : int
+increaseScore(score : int) : void
+decreaseScore(score : int) : void
+freezeEnemy() : void
1

Defenders' object and class diagram is shown in the figures above. Second and third figures are split versions of first figure. These classes are focused on only in game object-oriented design. In other words, graphical user interface, menu, settings class diagrams 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 functions except getters, setters can be found below. Firstly, two major classes, which are Game class and Map class, are explained. These classes control the game in general and with help of abstraction one can partially understand how the game works. After these classes had been explained LocatableObject Class is explained which has a big importance for the design because almost all classes after this one, are related to this class. Lastly, rest of the classes are explained.

5.3.1 Game Class

This class has the overall control of the game as it can be understood by its name. It has map, radar, current numbers of enemies, score, and level as attributes.

Game class will get level as parameter in the constructer function and according to level value it will design the beginning of game with private "createMap" 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, Game class will control over the stages of game such as pause stage, resume stage, refreshing map, boss stage, and finally ending the level. In the normal flow of the game, which is not in pause, this class will constantly call refresh map method to order map to change according to current situation and according to the next situation the class will change scores and radar.

5.3.2 Map Class

This class is like a real-life like map. It represents the current situation of the game. For example, where the buildings, enemies, or our spacecraft are hold in as information in this class and constantly manipulated according to flow of the game. In other words, Map class controls the current situation of the game while Game class controls general stages of game with having control over Map class, deciding when it changes or not. Without Game class, an object of Map class wouldn't be meaningful, therefore, there is a composition relationship between these class.

Map class has most of the classes such as buildings, spacecraft, bullets, enemies, meteors, buffs, boss as attributes and these classes don't have any life-cycles except being attributes of Map class, thus, there are a lot of composition relationship between Map class and other classes as shown in the figures above.

Map class has current situation of the map as attribute as well and it is constructed according to the other attributes of the class, meaning that "currentMap" represents the current map of the game at related time. Another important attribute of this class is view reflecting the part of the map that will be displayed as coordinates. Other attributes are related to boss of the level and freeze buff.

As mentioned in Game class there will be a level parameter to initialize the game according to level. Map will be start according to this parameter as well. Other functions of Map class are related to manipulating attributes that are mentioned above.

5.3.3 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 that are displayed on the map. Hence, there will be a lot of classes that will inherit this class. According to this requirement there are 4 important attributes of this class which are location (x, y coordinate), hitbox size, unique ID number, and isDead attribute to check whether the object needs to be removed.

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 functions are move methods which change the coordinate of the object via Location class and collapse method which will remove the object when it is dead.

5.3.4 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), and destination. These attributes are common for simple enemies. Evolution of enemies are done by changing its attributes. This class has no important method other than attribute manipulation method.

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

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

5.3.5 Tier1, Tier2, Tier3

These classes are representation of enemies according to their levels and have common methods that are controlled by simpleEnemy interface such as wandering around, radar searching, setting a destination location, or evolving. Furthermore, Tier1 and Tier3 enemies shot bullets via FireBullet interface. Tier2 enemies are suicide planes, so they have extra attribute to store their impact radius. Tier3 enemies can divide into Tier1 and Tier2 enemies. Lastly, since this classes inherit Enemy class evolution of enemies within their tier is done by changing their attributes from Enemy class.

5.3.6 Spacecraft

This class represents the players on the map. It has attributes for gun features, velocity, health, buff slot, smart bomb magazine, battery for hyper jump, infinite armor buff. Spacecraft can fire bullets via fireBullet interface as tier 1, 2, 3 enemies.

This class has functions for using buffs such as freeze enemies, infinite armor for a small time and other functions to manipulate rest of the attributes.

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

5.3.7 Bullet

This class is a small class to represent bullets that are produced by fireBullet interface. Bullets have damage, velocity, and direction attributes and methods to manipulate these attributes. This class has hit method to interact with other objects of the map when they intersect.

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

5.3.8 Boss

This class is to create bosses according to their level. Different bosses have different levels and dialogues along with health attribute. These different dialogues are created according to the level parameter that comes from Game class. Bosses can fire Lasers, rockets or little bosses. These classes are like bullets has damages but they will be represented differently when graphical user interface part is considered.

Boss and other classes produced by Boss class which are Laser, Rocket, LittleBoss inherit LocatableObject class. Additionally, these classes have composition relationship with Map class.

5.3.9 Building (Abstract)

This is a simple class to represent buildings on the map. It has health attribute and has no important method.

This class inherits LocatableObject class. Different buildings such as AllyBuilding, EnemyBuilding 1 and 2 inherit this class. This class has a composition relationship with Map class.

5.3.10 AllyBuilding, EnemyBuilding1, EnemyBuilding2

These classes are specialized building classes for different types of buildings. AllyBuilding has calling for back up function. EnemyBuilding2 host unevolved simple enemies to evolve them. EnemyBuilding 1 and 2 produces unevolved and evolved simple enemies respectively according to their level attributes that comes from Game class.

5.3.11 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 has explode method to damage when it collapses to other objects on the map.

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

5.3.12 Buff

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

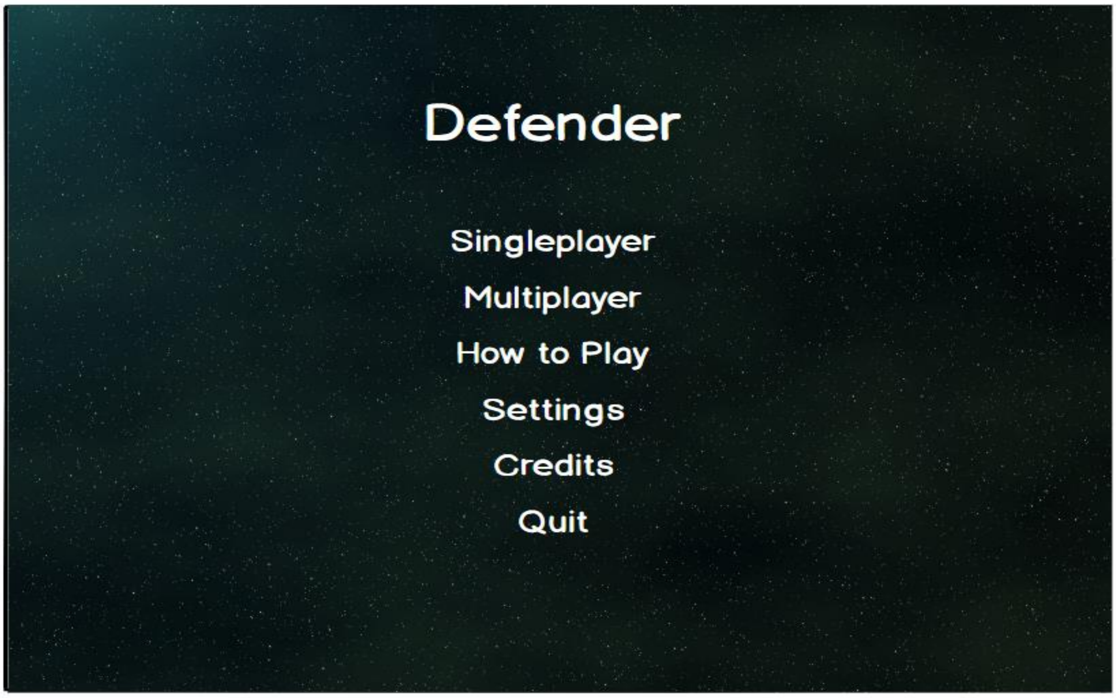
This class inherits LocatableObject class and has composition relationship with Map class.

5.3.13 Radar

This class represents the current situation of the map with Location objects. Game class has this class as attribute and manipulates it according to Map which is other attribute of Game class. Hence, there is a composition relationship between Radar and Game classes.

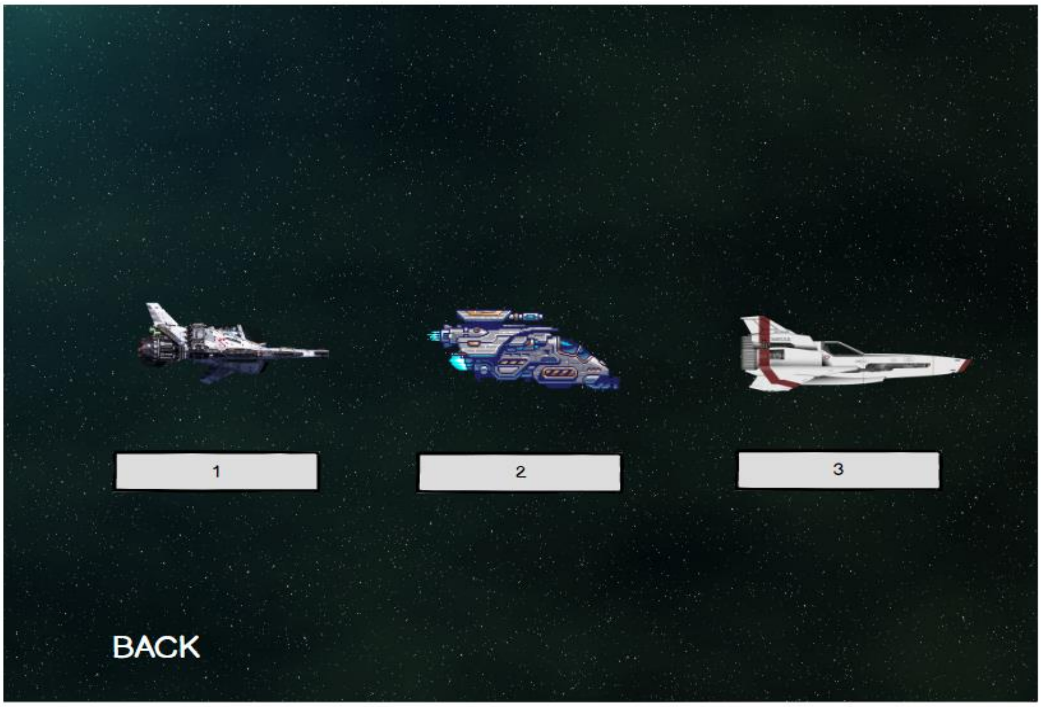
5.4 Screen Mock-ups

5.4.1 Main Menu



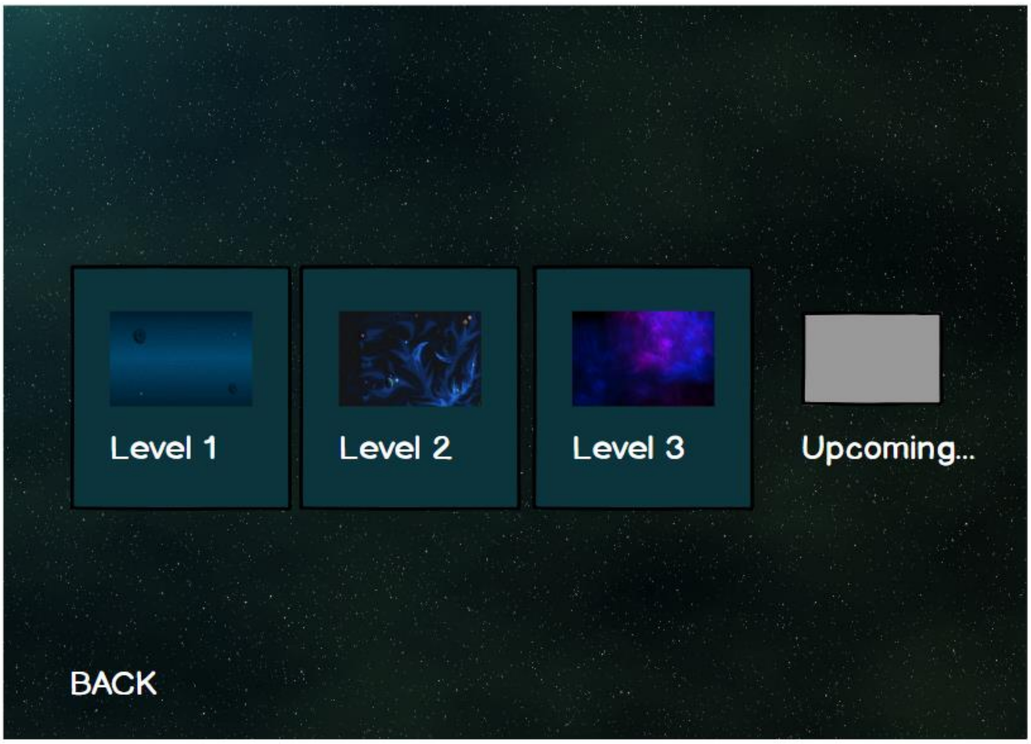
Main menu is the first screen of the game that contains 6 buttons. “Single Player” starts the game with one spaceship while “Two Player” starts the game with two spaceships. “How to Play” provides information about the game. “Settings” enables player to arrange volume and choose which keys will be used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

5.4.2 Spaceship Options



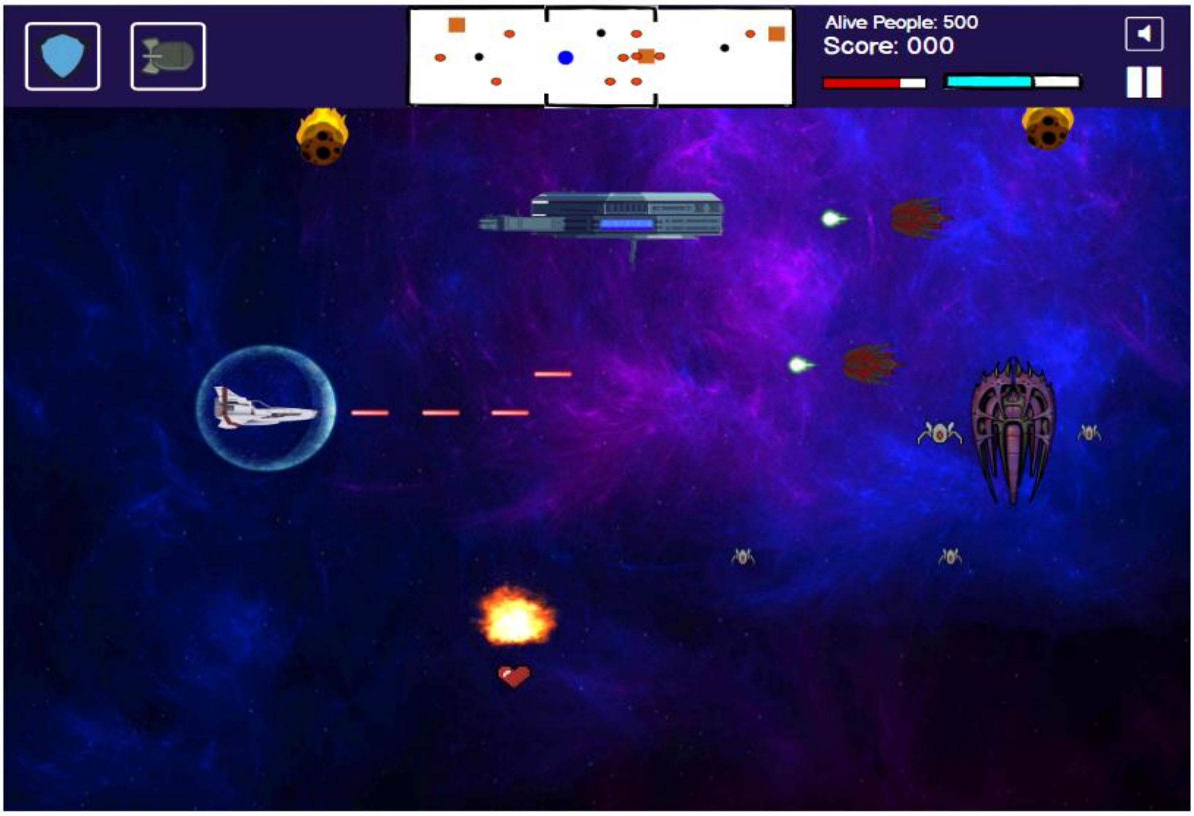
This screen helps the player to choose which spaceship s/he wants to use in the game. This screen contains a “Back” button to return main menu.

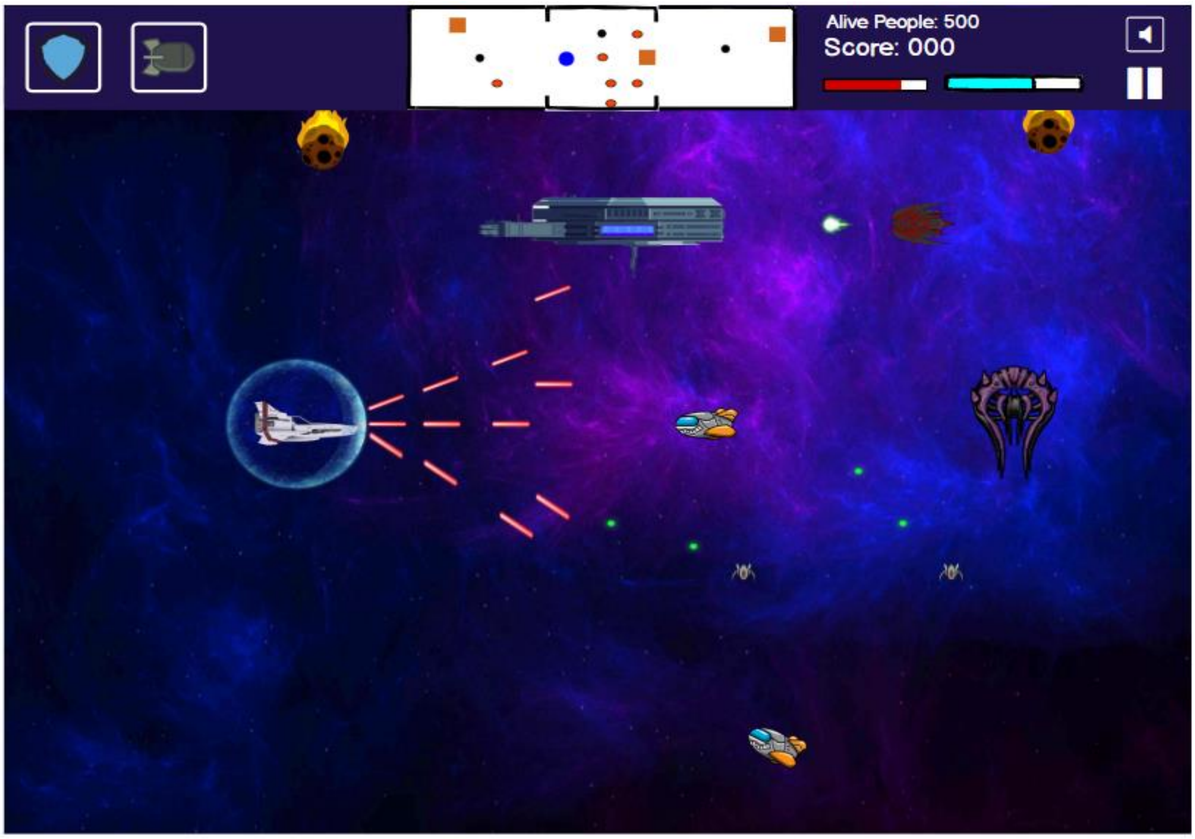
5.4.3 Level Options



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.

5.4.4 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 score information, number of alive people and slots for health points, smart bombs and hyperjump. A radar shows the place of enemies, enemy space stations and alliance 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.

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, 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. 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.