Bilkent University

Department of Computer Engineering

CS319 Term Project

Defender: Fiyuv++

Final Report

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

Instructor: Eray TÜZÜN

Teaching Assistants: Barış ARDIÇ, Alperen ÇETİN, Hasan BALCI

Contents

Final Report

Defender: Fiyuv ++

1. Introduction

We have divided our game into three parts, and we made work allocation according to these parts. We will first explain what is done for each part and then we will explain work done by each person.

For the menu part of the project:

- Three menu themes are provided for the player.
- Main menu, selection menus, high score, credits, and how to play views are designed. User selections in selection menus are taken and arranged to use in the game.
- Settings for key selection, volume and theme selection is designed and the game is arranged to use saved settings.
- All transitions between scenes are provided. Initialization of the game is done. Transition between levels is provided.
- Saving and loading settings, game information, game objects, and high scores are done in this part. We are able to save game in order to let player continue later.

In pre-boss scene:

- Levels are added into the game. In each level, a new enemy time is introduced into the game and spawning frequencies of enemy stations differ for each level.
- Three types of enemies are added with their behavior type. Their evolved versions are also added. Namely, we have six different enemy spacecrafts in the game.
- Simple enemy stations and evolved enemy stations are added into the game. They can spawn enemies. In level 1, they spawn Tier 1 enemy. In level 2, they spawn Tier 2 enemy and in level 3 they spawn Tier 1, 2 and 3 enemy.
- Buffs are added into the game.
- Hyper jump and smart bombs are added.
- Meteors are added into the game.
- Score calculation is added.
- Radar is added.

In this part, we implemented what we explained in design report except shield buff.

In boss scene:

- Three types of bosses are added into the game. Each boss has different type of behavior and each one of them has a special ability.
- BossOne throws laser as special ability.
- BossTwo throws rockets as special ability.
- BossThree throws little bosses as special ability.
- Buffs are added into this part of the game, too.

As we mentioned, work allocation is done throughout the semester according to these parts.

- We brainstormed and made decisions about the game in general together.
- [Doğukan, Aybüke] designed and implemented menu section of the game.
- [Musab] designed pre-boss section of the game and arranged how to use assets in the game.
- [Doğukan, Musab] implemented pre-boss section of the game.
- [Hamza, Meryem] designed and implemented boss section of the game.
- [Doğukan] was responsible for task distributions and managing meetings as team leader.
- [Meryem] was responsible for arranging document templates in drive.

For more detailed information about work allocation throughout the semester, you can access the link for “Project Meetings” document from our GitHub page. All meetings we have done since the beginning of the semester and all task distributions for the reports are available there.

2. Design Changes

2.1 Pre-boss Game Design Changes

At the start, we hypothetically constructed our system models such as object class diagrams, sequence diagrams according to the functional requirements. However, as we started to implement the game: Fiyuv++, we made both minor and major changes in our system models because there were some inconsistencies and poorly chosen design features.

First and the last situation of the object class diagram are shown in the figures below. The biggest major change in between is that at the first diagram we tried to implement boss scene and pre boss scene together, hence, both entity classes of pre boss scene and boss scene were using same map class and controller class. Since these two are separated considering most of the logic and classes such as Enemy, Building, Meteor, Buff, Boss, Laser, Rocket we decided to divide the class diagrams of these scenes into two and implement them by using different maps and controllers. Commonly used objects such as LocatableObject, Spacecraft, Bullet are shown in the both diagrams. We had also minor changes mostly in classes we defined at first diagram. The common biggest reason of these changes is the functions / attributes we just poorly placed at the first design and absent functions / attributes which is necessary to do the business logic.

When we start to implement the project, we figured out better approaches for some problems. For example, for the fire a bullet we decided to implement Strategy Design Pattern. With this way instead of writing a fireBullet function to every classes repeatedly, we can use one fireBullet function in multiple classes. Another example is that for controlling the flow of the game, we decided to group some functionalities and separate these functionalities into different classes. With this way, instead of having a one big controller class with lots of mission, we have game controller class to provide the connection between user inputs, the model of the game and the view of the game; we have spacecraft controller class to handle spacecraft related business logic; and we have map controller class to handle map related business logic. Other example of the design changes we made was about producing new enemies. We noticed that producing new enemies is a common feature in stations and we decided to combine this logic by applying Factory Design Pattern which will provide the enemy producing functionality.

It is very clear in the figures below that first version of our design decisions and later version of our design decisions have a lot of differences.

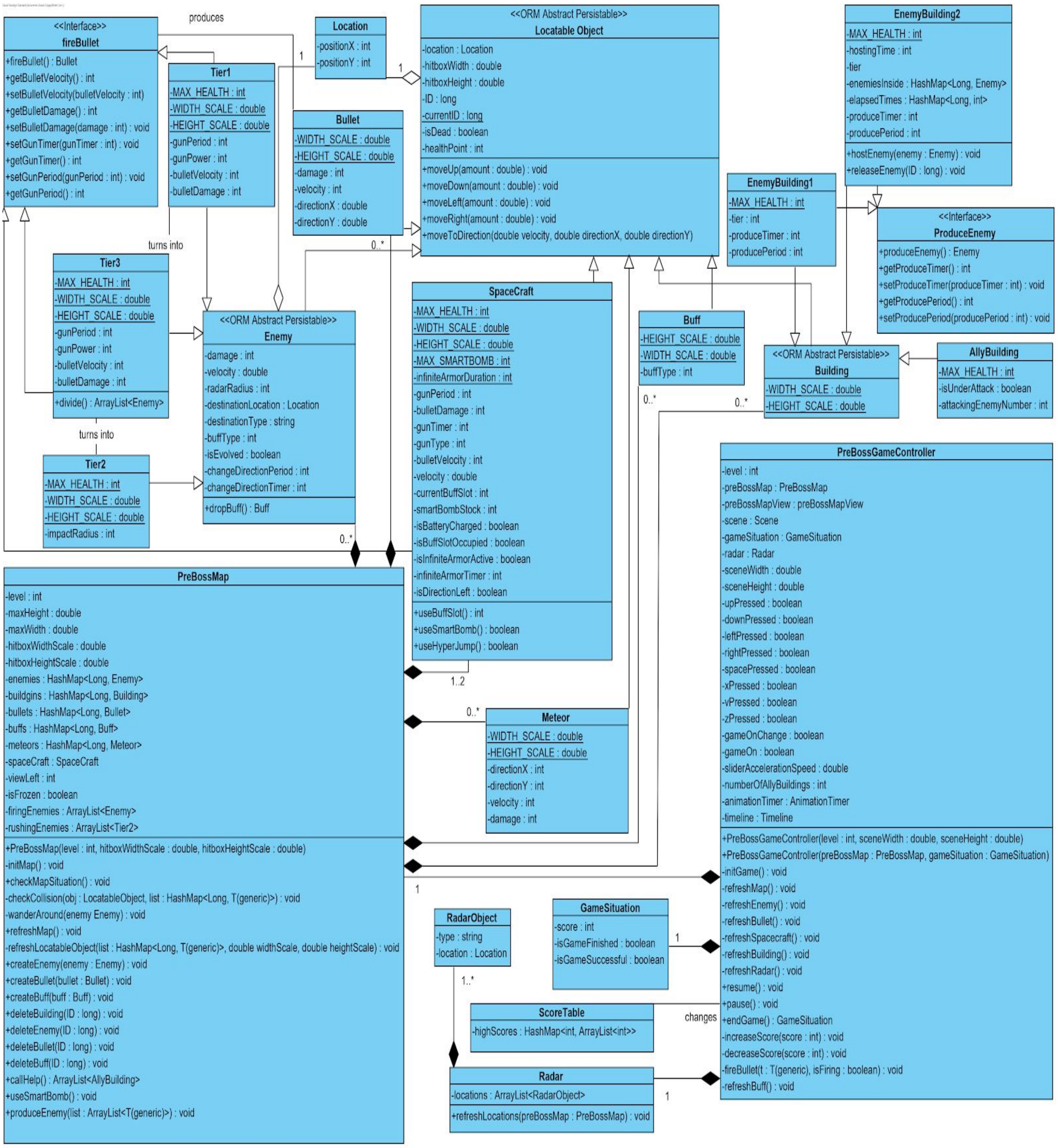

produces
<<Interface>>
fireBullet
+fireBullet() : Bullet
+getBulletVelocity() : int
+setBulletVelocity(bulletVelocity : int)
+getBulletDamage() : int
+setBulletDamage(damage : int) : void
+setGunTimer(gunTimer : int) : void
+getGunTimer() : int
+setGunPeriod(gunPeriod : int) : void
+getGunPeriod() : int
Tier1
-MAX_HEALTH : int
-WIDTH_SCALE : double
-HEIGHT_SCALE : double
-gunPeriod : int
-gunPower : int
-bulletVelocity : int
-bulletDamage : int
Location
-positionX : int
-positionY : int
1
1
Bullet
-WIDTH_SCALE : double
-HEIGHT_SCALE : double
-damage : int
-velocity : int
-directionX : double
-directionY : double
<<ORM Abstract Persistable>>
Locatable Object
-location : Location
-hitboxWidth : double
-hitboxHeight : double
-ID : long
-currentID : 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)
EnemyBuilding2
-MAX_HEALTH : int
-hostingTime : int
-tier
-enemiesInside : HashMap<Long, Enemy>
-elapsedTimes : HashMap<Long, int>
-produceTimer : int
-producePeriod : int
+hostEnemy(enemy : Enemy) : void
+releaseEnemy(ID : long) : void
EnemyBuilding1
-MAX_HEALTH : int
-tier : int
-produceTimer : int
-producePeriod : int
<<Interface>>
ProduceEnemy
+produceEnemy() : Enemy
+getProduceTimer() : int
+setProduceTimer(produceTimer : int) : void
+getProducePeriod() : int
+setProducePeriod(producePeriod : int) : void
turns into
0..*
Tier3
-MAX_HEALTH : int
-WIDTH_SCALE : double
-HEIGHT_SCALE : double
-gunPeriod : int
-gunPower : int
-bulletVelocity : int
-bulletDamage : int
+divide() : ArrayList<Enemy>
<<ORM Abstract Persistable>>
Enemy
-damage : int
-velocity : double
-radarRadius : int
-destinationLocation : Location
-destinationType : string
-buffType : int
-isEvolved : boolean
-changeDirectionPeriod : int
-changeDirectionTimer : int
+dropBuff() : Buff
SpaceCraft
-MAX_HEALTH : int
-WIDTH_SCALE : double
-HEIGHT_SCALE : double
-MAX_SMARTBOMB : int
-infiniteArmorDuration : int
-gunPeriod : int
-bulletDamage : int
-gunTimer : int
-gunType : int
-bulletVelocity : int
-velocity : double
-currentBuffSlot : int
-smartBombStock : int
-isBatteryCharged : boolean
-isBuffSlotOccupied : boolean
-isInfiniteArmorActive : boolean
-infiniteArmorTimer : int
-isDirectionLeft : boolean
+useBuffSlot() : int
+useSmartBomb() : boolean
+useHyperJump() : boolean
Buff
-HEIGHT_SCALE : double
-WIDTH_SCALE : double
-buffType : int
<<ORM Abstract Persistable>>
Building
-WIDTH_SCALE : double
-HEIGHT_SCALE : double
AllyBuilding
-MAX_HEALTH : int
-isUnderAttack : boolean
-attackingEnemyNumber : int
0..*
0..*
turns into
Tier2
-MAX_HEALTH : int
-WIDTH_SCALE : double
-HEIGHT_SCALE : double
-impactRadius : int
0..*
PreBossGameController
-level : int
-preBossMap : PreBossMap
-preBossMapView : preBossMapView
-scene : Scene
-gameSituation : GameSituation
-radar : Radar
-sceneWidth : double
-sceneHeight : double
-upPressed : boolean
-downPressed : boolean
-leftPressed : boolean
-rightPressed : boolean
-spacePressed : boolean
-xPressed : boolean
-vPressed : boolean
-zPressed : boolean
-gameOnChange : boolean
-gameOn : boolean
-sliderAccelerationSpeed : double
-numberOfAllyBuildings : int
-animationTimer : AnimationTimer
-timeline : Timeline
+PreBossGameController(level : int, sceneWidth : double, sceneHeight : double)
+PreBossGameController(preBossMap : PreBossMap, gameSituation : GameSituation)
-initGame() : void
-refreshMap() : void
-refreshEnemy() : void
-refreshBullet() : void
-refreshSpacecraft() : void
-refreshBuilding() : void
-refreshRadar() : void
+resume() : void
+pause() : void
+endGame() : GameSituation
-increaseScore(score : int) : void
-decreaseScore(score : int) : void
-fireBullet(t : T(generic), isFiring : boolean) : void
-refreshBuff() : void
PreBossMap
-level : int
-maxHeight : double
-maxWidth : double
-hitboxWidthScale : double
-hitboxHeightScale : double
-enemies : HashMap<Long, Enemy>
-buildgins : HashMap<Long, Building>
-bullets : HashMap<Long, Bullet>
-buffs : HashMap<Long, Buff>
-meteors : HashMap<Long, Meteor>
-spaceCraft : SpaceCraft
-viewLeft : int
-isFrozen : boolean
-firingEnemies : ArrayList<Enemy>
-rushingEnemies : ArrayList<Tier2>
+PreBossMap(level : int, hitboxWidthScale : double, hitboxHeightScale : double)
-initMap() : void
+checkMapSituation() : void
-checkCollision(obj : LocatableObject, list : HashMap<Long, T(generic)>) : void
-wanderAround(enemy Enemy) : void
+refreshMap() : void
-refreshLocatableObject(list : HashMap<Long, T(generic)>, double widthScale, double heightScale) : void
+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
+callHelp() : ArrayList<AllyBuilding>
+useSmartBomb() : void
+produceEnemy(list : ArrayList<T(generic)>) : void
1..2
0..*
Meteor
-WIDTH_SCALE : double
-HEIGHT_SCALE : double
-directionX : int
-directionY : int
-velocity : int
-damage : int
1
RadarObject
-type : string
-location : Location
GameSituation
-score : int
-isGameFinished : boolean
-isGameSuccessful : boolean
1
1..*
ScoreTable
-highScores : HashMap<int, ArrayList<int>>
changes
Radar
-locations : ArrayList<RadarObject>
+refreshLocations(preBossMap : PreBossMap) : void
1

Figure 2: Earlier Version of Object Class Diagram for Pre Boss Scene

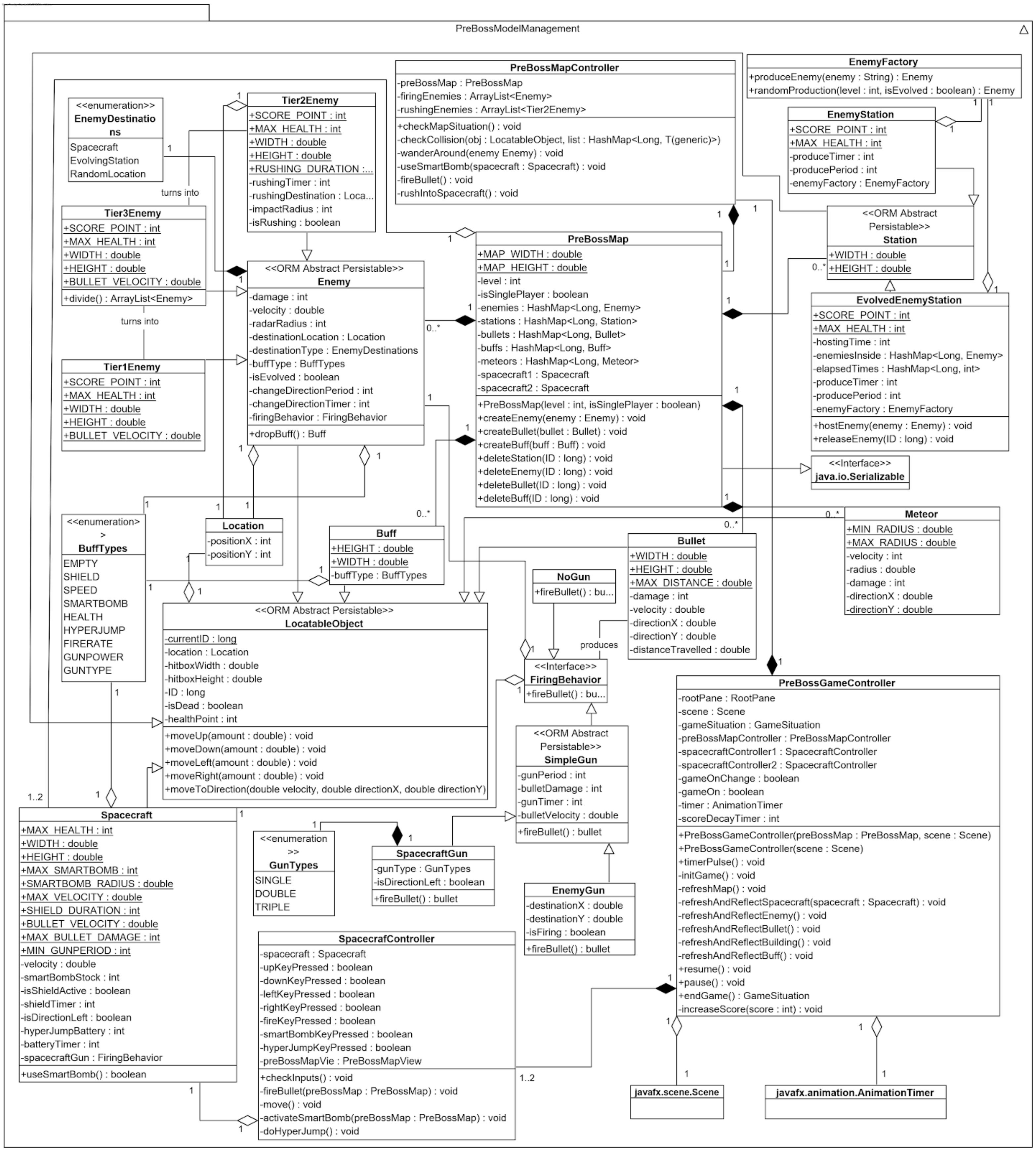
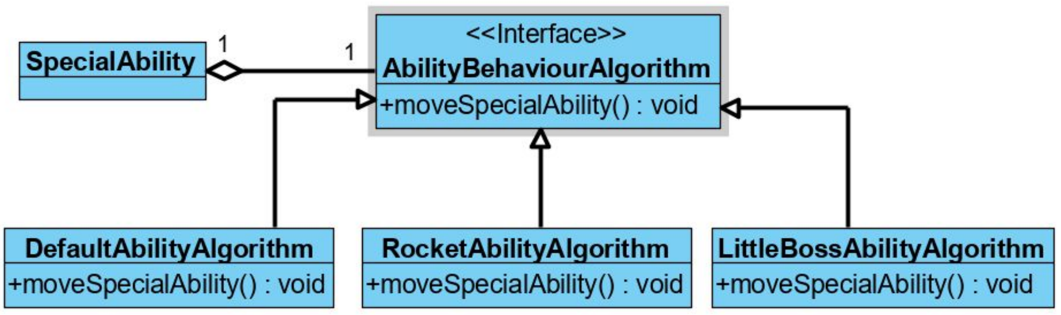

PreBossModelManagement
<<enumeration>> EnemyDestinations
Spacecraft
EvolvingStation
RandomLocation
Tier2Enemy
+SCORE_POINT : int
+MAX_HEALTH : int
+WIDTH : double
+HEIGHT : double
+RUSHING_DURATION :...
-rushingTimer : int
-rushingDestination : Loca...
-impactRadius : int
-isRushing : boolean
PreBossMapController
-preBossMap : PreBossMap
-firingEnemies : ArrayList<Enemy>
-rushingEnemies : ArrayList<Tier2Enemy>
+checkMapSituation() : void
-checkCollision(obj : LocatableObject, list : HashMap<Long, T(generic)>)
-wanderAround(enemy Enemy) : void
-useSmartBomb(spacecraft : Spacecraft) : void
-fireBullet() : void
-rushIntoSpacecraft() : void
EnemyFactory
+produceEnemy(enemy : String) : Enemy
+randomProduction(level : int, isEvolved : boolean) : Enemy
EnemyStation
+SCORE_POINT : int
+MAX_HEALTH : int
-produceTimer : int
-producePeriod : int
-enemyFactory : EnemyFactory
turns into
Tier3Enemy
+SCORE_POINT : int
+MAX_HEALTH : int
+WIDTH : double
+HEIGHT : double
+BULLET_VELOCITY : double
+divide() : ArrayList<Enemy>
<<ORM Abstract Persistable>> Enemy
-damage : int
-velocity : double
-radarRadius : int
-destinationLocation : Location
-destinationType : EnemyDestinations
-buffType : BuffTypes
-isEvolved : boolean
-changeDirectionPeriod : int
-changeDirectionTimer : int
-firingBehavior : FiringBehavior
+dropBuff() : Buff
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
+deleteStation(ID : long) : void
+deleteEnemy(ID : long) : void
+deleteBullet(ID : long) : void
+deleteBuff(ID : long) : void
<<ORM Abstract Persistable>> Station
+WIDTH : double
+HEIGHT : double
EvolvedEnemyStation
+SCORE_POINT : int
+MAX_HEALTH : int
-hostingTime : int
-enemiesInside : HashMap<Long, Enemy>
-elapsedTimes : HashMap<Long, int>
-produceTimer : int
-producePeriod : int
-enemyFactory : EnemyFactory
+hostEnemy(enemy : Enemy) : void
+releaseEnemy(ID : long) : void
<<Interface>> java.io.Serializable
turns into
Tier1Enemy
+SCORE_POINT : int
+MAX_HEALTH : int
+WIDTH : double
+HEIGHT : double
+BULLET_VELOCITY : double
<<enumeration>> BuffTypes
EMPTY
SHIELD
SPEED
SMARTBOMB
HEALTH
HYPERJUMP
FIRERATE
GUNPOWER
GUNTYPE
Location
-positionX : int
-positionY : int
Buff
+HEIGHT : double
+WIDTH : double
-buffType : BuffTypes
Meteor
+MIN_RADIUS : double
+MAX_RADIUS : double
-velocity : int
-radius : double
-damage : int
-directionX : double
-directionY : double
Bullet
+WIDTH : double
+HEIGHT : double
+MAX_DISTANCE : double
-damage : int
-velocity : double
-directionX : double
-directionY : double
-distanceTravelled : double
NoGun
+fireBullet() : bu...
produces
<<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)
<<Interface>> FiringBehavior
+fireBullet() : bu...
<<ORM Abstract Persistable>> SimpleGun
-gunPeriod : int
-bulletDamage : int
-gunTimer : int
-bulletVelocity : double
+fireBullet() : bullet
PreBossGameController
-rootPane : RootPane
-scene : Scene
-gameSituation : GameSituation
-preBossMapController : PreBossMapController
-spacecraftController1 : SpacecraftController
-spacecraftController2 : SpacecraftController
-gameOnChange : boolean
-gameOn : boolean
-timer : AnimationTimer
-scoreDecayTimer : int
+PreBossGameController(preBossMap : PreBossMap, scene : Scene)
+PreBossGameController(scene : Scene)
+timerPulse() : void
-initGame() : void
-refreshMap() : void
-refreshAndReflectSpacecraft(spacecraft : Spacecraft) : void
-refreshAndReflectEnemy() : void
-refreshAndReflectBullet() : void
-refreshAndReflectBuilding() : void
-refreshAndReflectBuff() : void
+resume() : void
+pause() : void
+endGame() : GameSituation
-increaseScore(score : int) : void
Spacecraft
+MAX_HEALTH : int
+WIDTH : double
+HEIGHT : double
+MAX_SMARTBOMB : int
+SMARTBOMB_RADIUS : double
+MAX_VELOCITY : double
+SHIELD_DURATION : int
+BULLET_VELOCITY : double
+MAX_BULLET_DAMAGE : int
+MIN_GUNPERIOD : int
-velocity : double
-smartBombStock : int
-isShieldActive : boolean
-shieldTimer : int
-isDirectionLeft : boolean
-hyperJumpBattery : int
-batteryTimer : int
-spacecraftGun : FiringBehavior
+useSmartBomb() : boolean
<<enumeration>> GunTypes
SINGLE
DOUBLE
TRIPLE
SpacecraftGun
-gunType : GunTypes
-isDirectionLeft : boolean
+fireBullet() : bullet
EnemyGun
-destinationX : double
-destinationY : double
-isFiring : boolean
+fireBullet() : bullet
SpacecrafController
-spacecraft : Spacecraft
-upKeyPressed : boolean
-downKeyPressed : boolean
-leftKeyPressed : boolean
-rightKeyPressed : boolean
-fireKeyPressed : boolean
-smartBombKeyPressed : boolean
-hyperJumpKeyPressed : boolean
-preBossMapVie : PreBossMapView
+checkInputs() : void
-fireBullet(preBossMap : PreBossMap) : void
-move() : void
-activateSmartBomb(preBossMap : PreBossMap) : void
-doHyperJump() : void
javafx.scene.Scene
javafx.animation.AnimationTimer
1..2
0..*
1

Figure 2: Changed Version of Object Class Diagram for Pre Boss Scene

2.2. Boss Game Design Changes

Firstly, in the Boss Game, each special ability is moving in the Boss Map differently. For instance, Little Boss ability should bounce when it hits the boundaries, but Rocket ability should travel until it reaches its corresponding marker. To find a solution to this changing factor, we introduced AbilityBehaviourAlgorithm interface and we decided to apply Strategy Design Pattern.


SpecialAbility
1
1
<<Interface>>
AbilityBehaviourAlgorithm
+moveSpecialAbility() : void
DefaultAbilityAlgorithm
+moveSpecialAbility() : void
RocketAbilityAlgorithm
+moveSpecialAbility() : void
LittleBossAbilityAlgorithm
+moveSpecialAbility() : void

Secondly, it was decided to apply Singleton Design Pattern to BossMap class, but we decided not to. Because it is realized that having one object and being a Singleton class are different things.

2.3. Menu Design Changes

- While preparing diagrams for design report, we were not planning to use CSS. However, we recognized that preparing themes for menu will be much easier when we use them. Therefore, we added css files for the views of the menu.

- We added a few extra classes for menu views under the heading of “menu entities” to be able to design game menus easily.

3. Lessons Learnt

We learnt a lot of lessons during the project. Some of these lessons are about technical issues related to implementation of the project which includes concepts about IntelliJ, Java, Javafx, GitHub, Maven. Other lessons are about less technical, more documentation and design related issues about analysis and design report.

To begin with, none of us had any experience with Javafx and some of the group members didn’t have any experience with IntelliJ, GitHub, Maven. Moreover, we needed to brush up our memories with Java to remember what Java offers to developers such as inheritance, polymorphism, encapsulation, abstraction, generics, serialization, collections. Therefore, all members learned what is necessary for implementation of the project and during this process we gain many experiences. For example, considering Javafx, we learned how to control classes and object by using AnimationTimer, TimeLine, or ChangeListener. As we spent time using and learning Javafx, we noticed some utilization that offers such as Javafx properties or Javafx Collections, i.e. instead of using Boolean we used BooleanProperty or instead of using ArrayList we could use ObservableList. These utilizations use Java, but in an optimized way for JavaFx. Some of these utilizations, we figured earlier in implementation process and used. For some others, we figured a little later, so we needed to evaluate that how much we needed to spend to change our implementation. According to the evaluation we decide whether we will use the utilization or not. Lesson that we learn from this could be stated as before jumping into implementation, comprehensively investigating the tools for usage would be better approach. Another example of this lesson was when we spend almost 2 hours to understand why we cannot access to the classes imported by Maven, which was because we needed to require the classes for configuration after importing by Maven.

Secondly, we also didn’t have any experience with design and documentation of a software. However, we needed to immediately jump into design and documentation processes. Thus, we made a lot of mistakes at the beginning. As we learned from our mistakes and studied to the lectures, we tried to correct these mistakes. One example of this mistakes was to ignore talking about small details such as class names, as if we already had consensus on them or we could discuss them later. Ignoring the small details lead to inconsistency in both documentation of reports and designing the diagrams. We didn’t even had consistency about our game name. Hence, the decisions about small details of the software shouldn’t be ignored or postponed.

Thirdly, we learned that spending time on how to optimize our energy and decide our strategy instead of immediately jumping into a task is more efficient. This approach can save a lot of times and unnecessary labor that might happen in the future. luckily, after first iterations we adopted this strategy and it helped us a lot in the second iterations.

Lastly, one of the biggest lessons we learn in our project experience is that when we encounter a problem about anything, it is very likely that somebody was faced the same problem before us and there is an optimal strategy we can apply to our situation. Design patterns was one of the examples of this situation. Not only low level problems but also other problems could be the examples of this golden lesson.

4. User's Guide

For the first player, default keys are the followings:

- Arrow keys for moving
- Numpad1 key for shooting
- Numpad 2 key for doing hyper-jump
- Numpad 3 key for using smart bomb.

For the second player - if the 2 players mode is chosen- default keys are the followings:

- W-A-S-D for moving
- Space key for shooting
- Z key for doing hyper-jump
- X key for using smart bomb

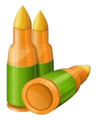
The player always can change the default keys from settings.

During the game, you can collect different bonuses. Here is the list of the bonuses and their benefits:

- Speed Bonus: The spacecraft moves faster.
- Smart Bomb Bonus: The spacecraft gains one more smart bomb. Smart bomb destroys everything that is visible in the spacecraft's radar. Maximum number of smart bombs that the spacecraft can carry is 3.
- Health Point Bonus: Health point of spacecraft is increased.
- Hyper-Jump Bonus: It refills the spacecraft's hyper-jump charge. Spacecraft can teleport forward proportional to its hyper-jump charge. The spacecraft cannot use hyper-jump ability if its charge is below 25%. Other way to fill up hyper jump is to wait. For every second hyper-jump charge is increased.
- Fire Rate Bonus: The spacecraft shoots faster.

- Gun Power Bonus: The spacecrafts hits harder.

- Gun Type Bonus: The spacecraft gains multishot ability.



There are also different enemies. The player should kill all of them to pass the level.

- Tier 1 Enemy: The enemy is simple. When it sees the player it starts shooting. But don't afraid because it can see you only from short distances.

- Tier 2 Enemy: The enemy tries to crush into you like a kamikaze.

- Tier 3 Enemy: It shoots from long distances. When the player kills it it splits into 2: Tier 1 enemy and Tier 2 enemy.

The enemies can be in evolved from. Enemies in evolved form are improved by increasing their health point, attack rate etc.

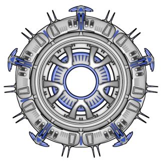
- Evolved Tier 1 Enemy: It has the same behaviour with Tier 1 Enemy.

- Evolved Tier 2 Enemy: It has the same behaviour with Tier 2 Enemy.

- Evolved Tier 3 Enemy: It has the same behaviour with Tier 3 Enemy.

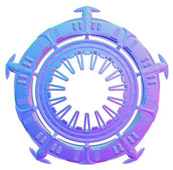
There are stations which can produce enemies. The player should destroy all of them to pass the level.

- Simple Enemy Station: Produces unevolved enemies.



The stations can be also in evolved form. In addition to producing enemies, it makes unevolved enemies evolved.

- Evolved Enemy Station: Produces evolved enemies. It can make unevolved enemies evolved.



Lastly, there are different bosses at the end of each level. They have different abilities to kill spacecraft. Good luck dealing with them!

5. Build Instructions

Since Fiyuv++ is a simple Java game, it doesn't need too many system requirements. Fiyuv++ has a exe file, so we expect that the game can work every desktop platform that has Java Runtime Environment 13.

To run Fiyuv++:

- You should install JRE 13 - [Download JRE13]()
- Your computer should have 250MB free space (200MB for JRE13, 30-40MB for game)
- After downloading Fiyuv++, you can play it by running Fiyuv++.exe file.