Bilkent University

Department of Computer Engineering

CS319 Term Project

Defender: Fiyuv++

Design Report

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

Instructor: Eray TÜZÜN
Teaching Assistant(s): Barış ARDIÇ, Alperen ÇETİN, Hasan BALCI

Contents

Design Report

Defender: Fiyuv ++

1. Introduction

1.1. Purpose of the System

The purpose of the system is to entertain a player through the satisfaction of challenge, survival, defense of humanity, and team work in two player mode. We will try to create a game with high performance and user-friendly interface to help players engage in our space world.

1.2. Design Goals

1.2.1. User-Friendly Interface

Fiyuv++ is designed to be played easily. A player has a chance to change keys for movement and fire. In this way, player will be comfortable with the keys they use throughout the game. Even if the player does not select any keys to use, the most frequently used keys in the games will be defined as default. Icons used for voice, pause and most of the bonuses will be similar to the frequently used icons for these purposes so that players can easily understand what is their purpose. Also, the game will have a color option for color-blind people. A ‘How to Play’ page which is accessible from main page will also help players get used to the game environment.

1.2.2. Performance

We will use JavaFX to have a good quality of performance. We chose JavaFX because of its superiority in the performance of graphics. We expect our game to have 30-60 fps depending on the system that runs the game. We also are planning to make the animations as smooth as possible.

1.2.3. Maintainability

Our game is designed using object-oriented programming concepts. Using class system, we can make small changes or add new qualities to the game without getting into trouble with bugs or complexity of system. Therefore, our system will be easily extensible and modifiable. Furthermore, we will use Java while implementing our game and it will enable people to play our game in environments with various operating and hardware systems.

1.2.4. Responsive Design

We designed our game so that the layout and orientation of the game changes according to player’s screen size. As the screen size differs from one device to another, we thought that it is important to provide user an appropriate layout to increase satisfaction that players get from the game.

1.2.5. Robustness

We will try to implement our game so that any misuse of functionalities will not cause a big failure and the system will tolerate this type of error.

Trade-offs

Portability vs Development Time

Providing portability is not an easy job. Sometimes, it may need rewriting some parts of the code. Therefore, we will only able to implement our game as a desktop application. The game will not be played in other platforms.

Development Time vs Maintainability

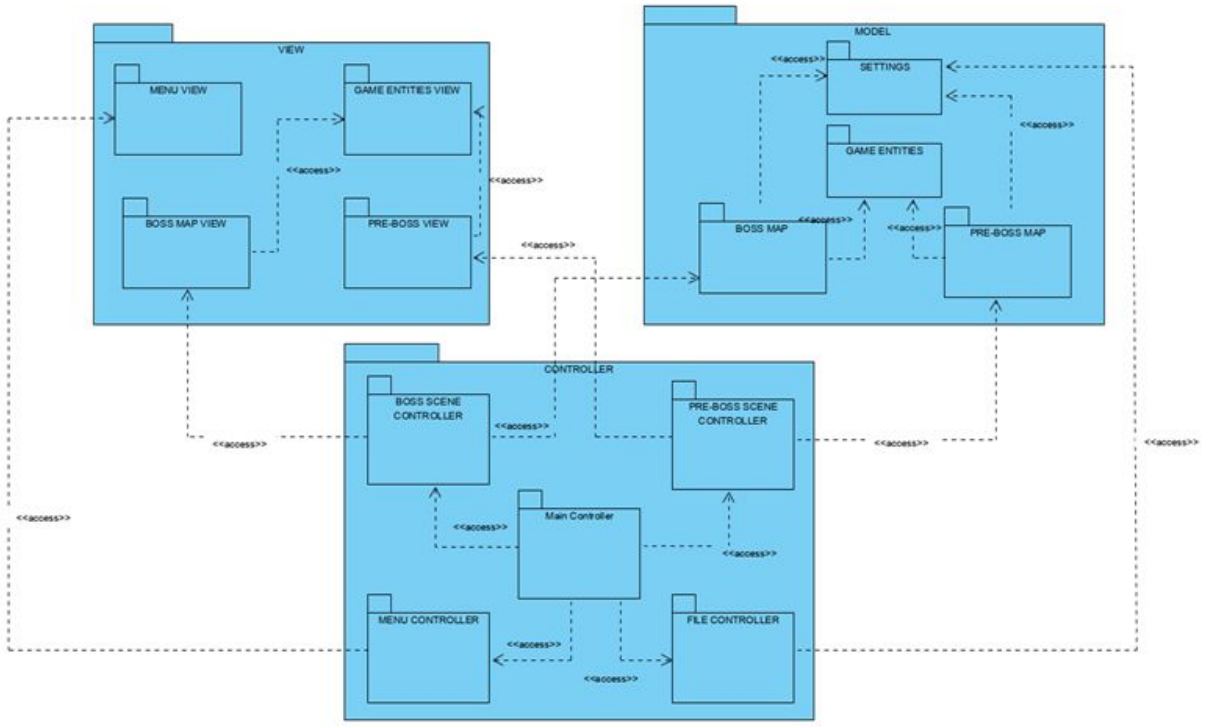
We used object-oriented programming principles to design our game. Although design process takes so much time, our game will be maintainable through class hierarchy.

Performance vs Memory

A game with a low performance would not be so entertaining. Therefore, we will try to improve game performance rather than focusing on the use of memory.

2. System Architecture

2.1. Subsystem Decomposition



In this section, we will divide our project into subsystems. Because, it is difficult to understand the whole system without divide it into pieces. Therefore, our purpose of subsystem decomposition is increasing the understandability and coherence of the system. We have decided to choose MVC (Model-View-Controller) structure as design pattern for our project. Then, we created our subsystems according to MVC structure.

We choose MVC design pattern because;

- It separates layers for the sake of simplicity so that business logic and UI are in different layers.
- Also, separation of layers enables us to focus on one aspect of the project at a time.
- Because of separation of layers, it enables that multiple people can work simultaneously on project.
- Its architecture increases the coherence.
- Its separated layers make future improvements of project easy.
- Implementation of the structure is not difficult.

Our MVC design has 3 main packages, these are View, Controller and Model as in MVC.

- Model package contains game entities, maps and settings sub packages. Game entities are basically all object that can be seen in the game. Maps contains these game objects and it contains methods for interaction of game object such as their collision are checked in map models. Controller package interacts with this package and it updates objects' data.
- Controller package is where business logic is implemented for the project. It is like a bridge between View and Model package. It controls Model package according to inputs coming from view and it sends updated data of model to view package so that view can be updated according to input.
- View contains the UI components of game of objects such us game entities views, map view, menu view. View package is where the user interacts with the game. One user can change the data of model through controller by

interacting with UI and View package request updated data from Model and it takes data from model through controller and it displays object according to updated data.

2.2. Hardware/Software Mapping

The game will be written and implemented in Java and we will use Javafx software platform which is a java GUI library. Javafx is not included in Java SDK, therefore it will be included as external libraries with the help of Maven. We will use Javafx 13 as version. In order for Javafx 13 to work properly it is required to have a recent version of JDK 13, or at least JDK 11. After deployment the game will work in every platform with related Java Runtime Environment installed.

As a hardware mapping, keyboard and mouse will be hardware requirement of the game. In play game situation keyboard will be used as input tool for spacecraft's movement and other features such as using buff or smartbomb or hyperjump, firing bullets and opening pause menu. Each of this functions will have specific key bindings that can be changed from the settings menu. Other than in play game situation and assigning key bindings, mouse will be used as input tool. Mouse will be used to click buttons, change settings, chose levels, spacecrafts before the game. Furthermore, non-volatile storage of the hardware will be used as input/output tool in order to achieve data persistency.

2.3. Persistent Data Management

Fiyuv++ game will have specific objects to have data persistency. For example, last preferred settings such as key bindings, level of volume, theme; current situation of the game in-play in case there will be error leading to crashing of the program, and high scores of the game needs to be persistent data that life-times won't end with the life-time of the game. In order to achieve data persistency we will use Serializable interface of Java. This interface enables objects to be **serialized** by converting its state to a byte stream so that the byte stream can be reverted back into a copy of the object by **deserialization**. To save serialized object we will use file system. We won't use database since we don't have multiple users or our data don't need to be ported across multiple platforms.

2.4. Access Control Security

The Defender game will not require any internet connection to play game. Therefore, we have not login page or authentication and authorization of users for the game. Also, high scores and user's settings will be stored in a text file so that nobody can access the stored data in the game except player itself. In return of no internet connection, highest scores will be unique for every different computer.

2.5. Boundary Conditions

2.5.1. Initialization

When the game initialized, it receives settings' data from text file. If user changes settings from menu, also these settings will be changed in the text file so that when the game is restarted, updated settings will be remain.

2.5.2. Termination

Termination of the game will be handled from different components. For instance, the game can be terminated from the main menu or from the pause menu. When a user terminates the game, its settings and last process in the level will be saved into the text file.

2.5.3. Failure

In the game, we will save the game's map automatically into the text file every 1 minutes against any crash that can be happen in the game. If game is crashed, user can start the game where it remains with at most 1 minutes loss.

3. Subsystem Services

3.1. User Interface Subsystem

View is where the user interacts with the system. User can see system response of an action through this package. Our game view consists of 4 components.

- **Menu View:** Menu view manages main menu of the game according to the commands of menu controller. It contains views of main menu such as how to play, settings and credits screens, and enables transition between them.
- **Game Entities View:** Game Entities View contains view classes of game entities such as bullet, spacecraft, building, meteor, enemies or bosses. Every game entity view class contains properties and methods to change the view of that entity in the map.
- **Pre-Boss Map View:** Pre-Boss Map View adds each entity into a hashmap with an ID and changes the appearance or positions of game entities according to the commands of Pre-Boss Scene Controller using their ID when the user interacts with pre-boss scene. This view can only change the objects that belongs to pre-boss map scene such as buildings, bullets, enemies or buffs.
- **Boss View:** Boss View has a similar construction with pre-boss map view and it refreshes map according to the commands of Boss Scene Controller when the user interacts with boss scene at the end of each level. This view can only change the objects that belongs to boss scene such as bosses or bullets.

3.2. Controller Subsystem

Controller Subsystem consists of controller classes and tackles the connection between game model and UI. In our game, there are 5 main controller components.

- **Main Controller:** Main Controller manages other controllers. It checks user's choices and according to them, directs the user to desired controller component. To give an example, if user wants to change settings, main controller forwards this task to file controller. Additionally, main controller manages the transition from the scene of game before boss to the scene of fighting with boss.
- **Menu Controller:** Menu Controller handles transitions of menu pages before the game starts.
- **File Controller:** To be consistent, our game saves the last settings which user changes, the last 10 high scores, and opened levels until the last time user played. File controller performs these saving and loading operations. When game is opened, file controller loads last settings. Similarly, it saves current situations before user exit the game.
- **Pre-Boss Scene Controller:** Pre-Boss Scene Controller is responsible for whole game process before the player faces with the boss. Firstly, this controller have a method which takes pressed key as input and gives this information to the player's spacecraft class to update spacecraft's location. Secondly, this controller takes the information about game entities from model and gives them to the view class by using refresh methods. These informations are game entities' locations and whether they are dead or not.

According to these informations, Pre-Boss Scene Controller gives an order to the view class to set their locations or remove dead ones from map.

- **Boss Scene Controller:** Boss-Scene Controller is responsible for the game part which the player fights with the boss. Similarly to Pre-Boss Scene Controller, this controller takes movements and attacks of both the boss and the player's spacecraft, and gives them to the view class to update map. Also, this controller manages the dialogues between the boss and the player.

3.3. Model Subsystem

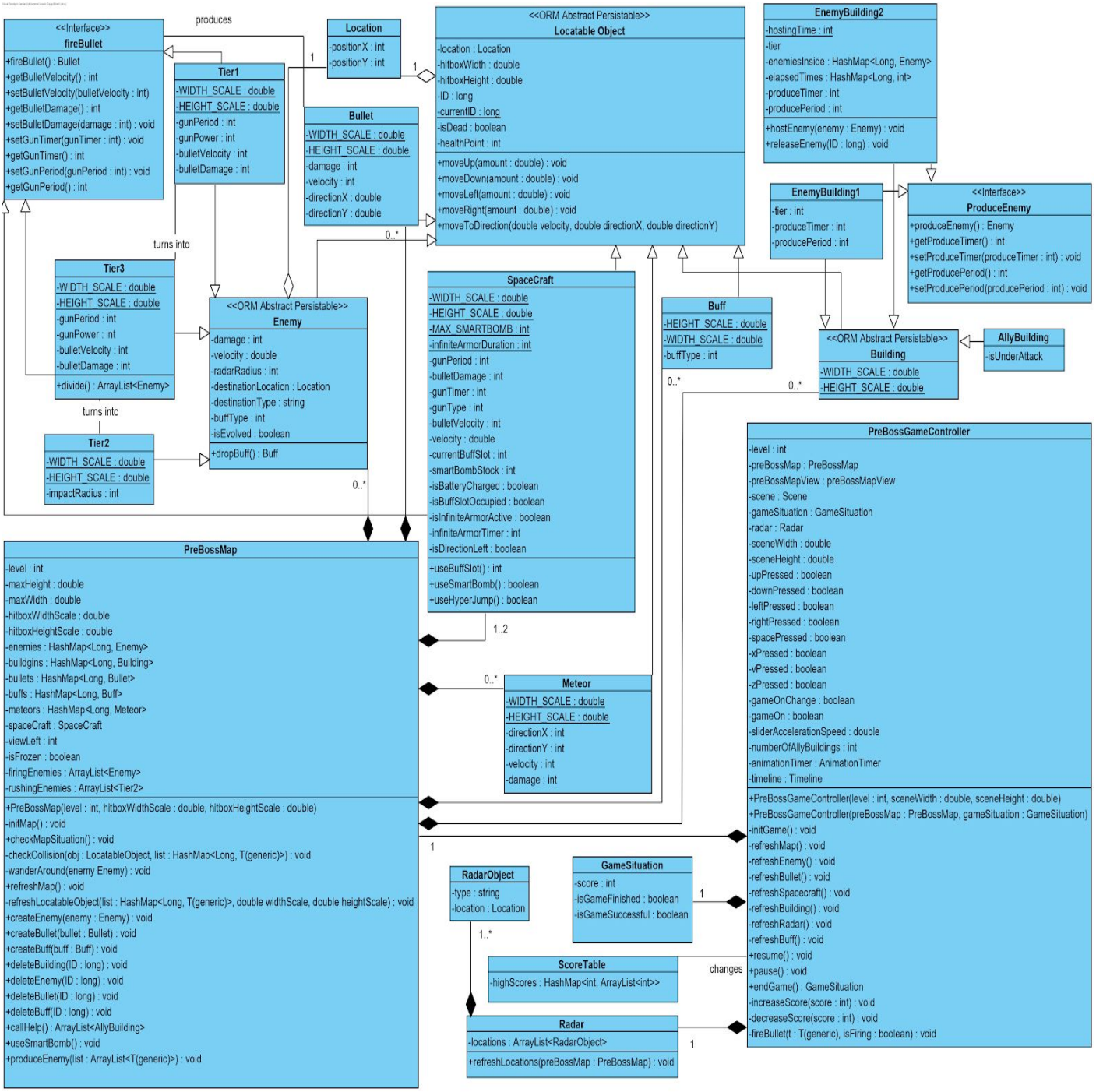
Model Subsystem can be defined as the algorithm behind the game. It consists of 4 main components:

- **Game Entities:** This component consists of model classes of game objects such as spacecraft, enemy tiers, bullets etc. Every entity model class includes properties and methods of that entity.
- **Pre-Boss Map:** Pre-Boss Map keeps lists of game entities. If a new entity is created, it adds this new entity to list. Similarly, if an entity is dead or destroyed, Pre-Boss Map removes this entities from the list. Additionally, Pre-Boss Map checks collision of game entities.
- **Boss Map:** Boss Map is similar to Pre-Boss Map. However, in boss scene, there is no other enemies.
- **Settings:** Settings component includes the necessary attributes about settings such as volume level, keys which is used by the player etc.

4. Low Level Design

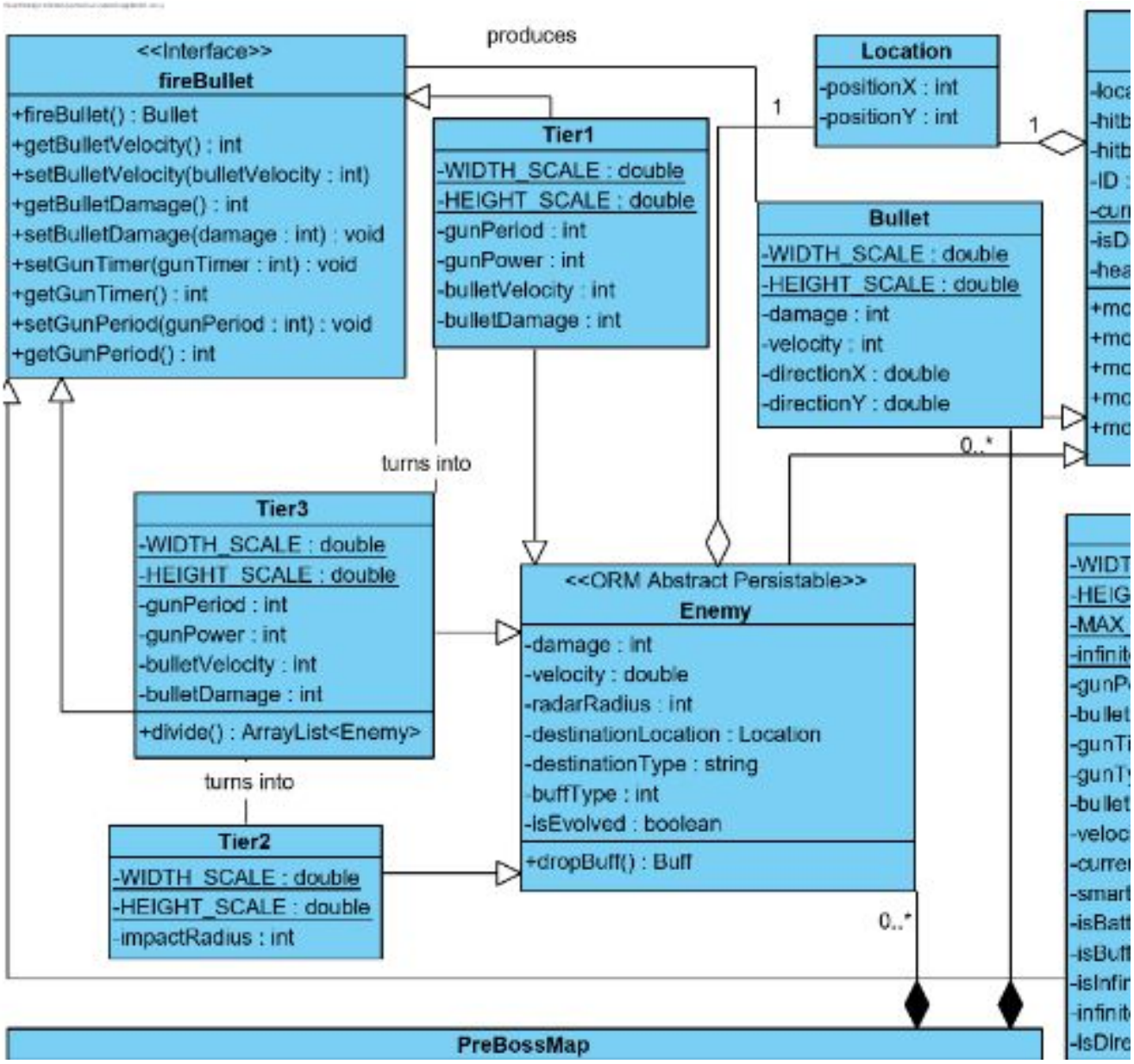
4.1. Pre Boss Class Interfaces

PreBoss Controller-Model Classes All Parts

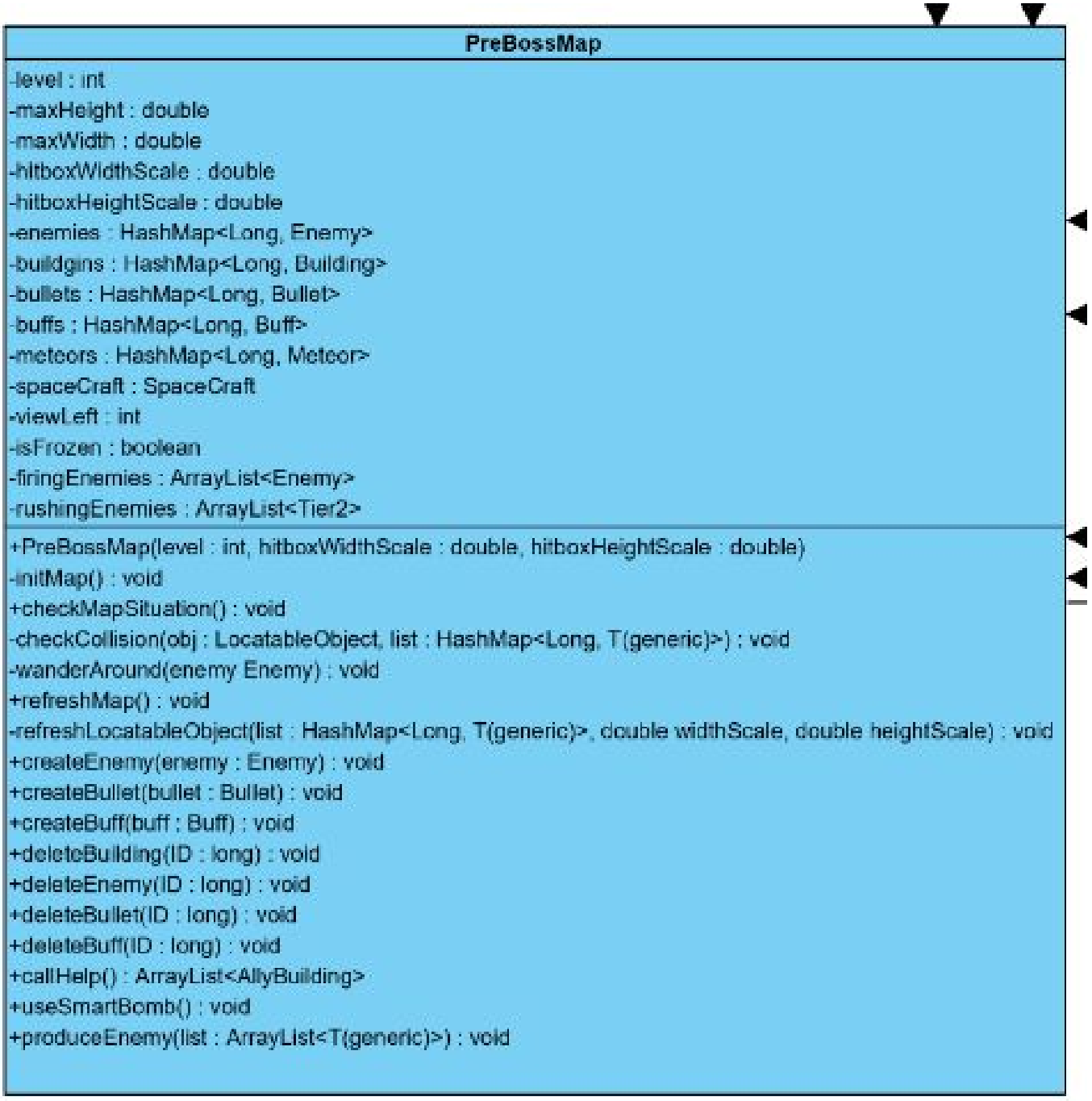


PreBoss Controller Model Classes Parts 1-5

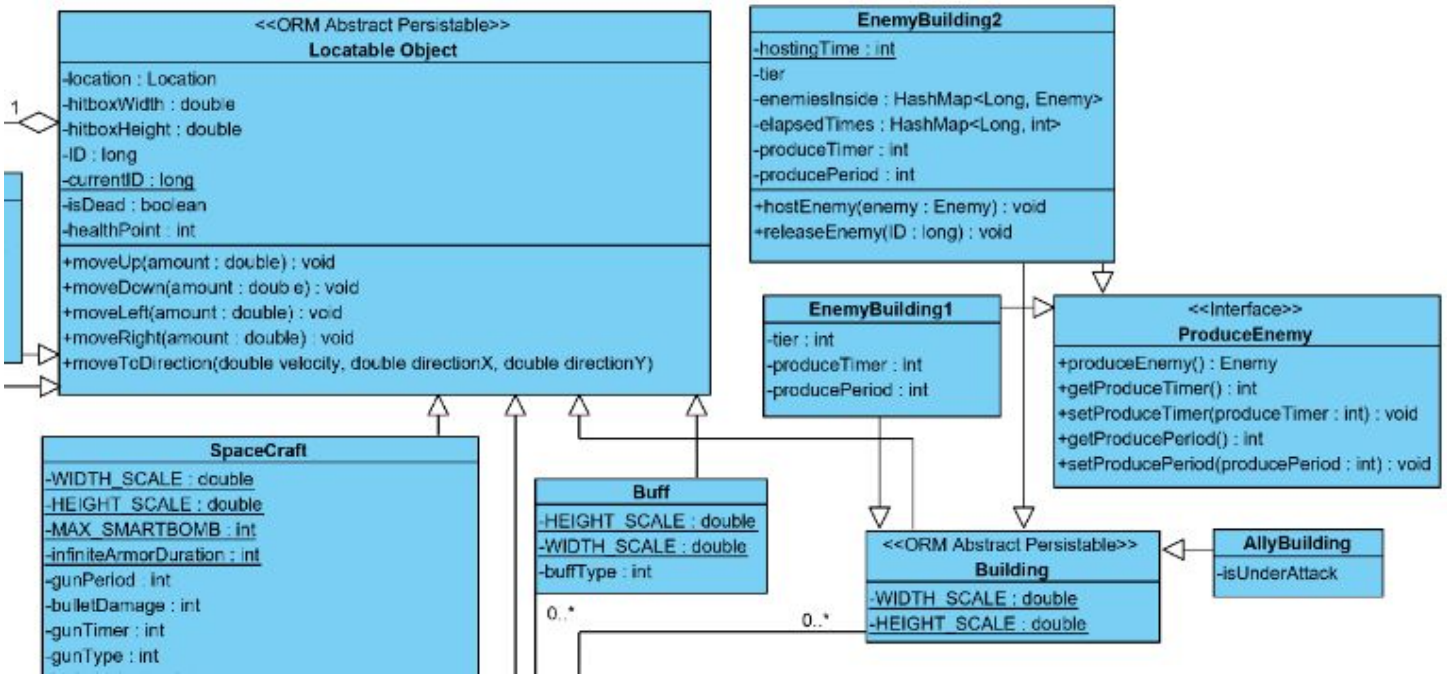
Part 1



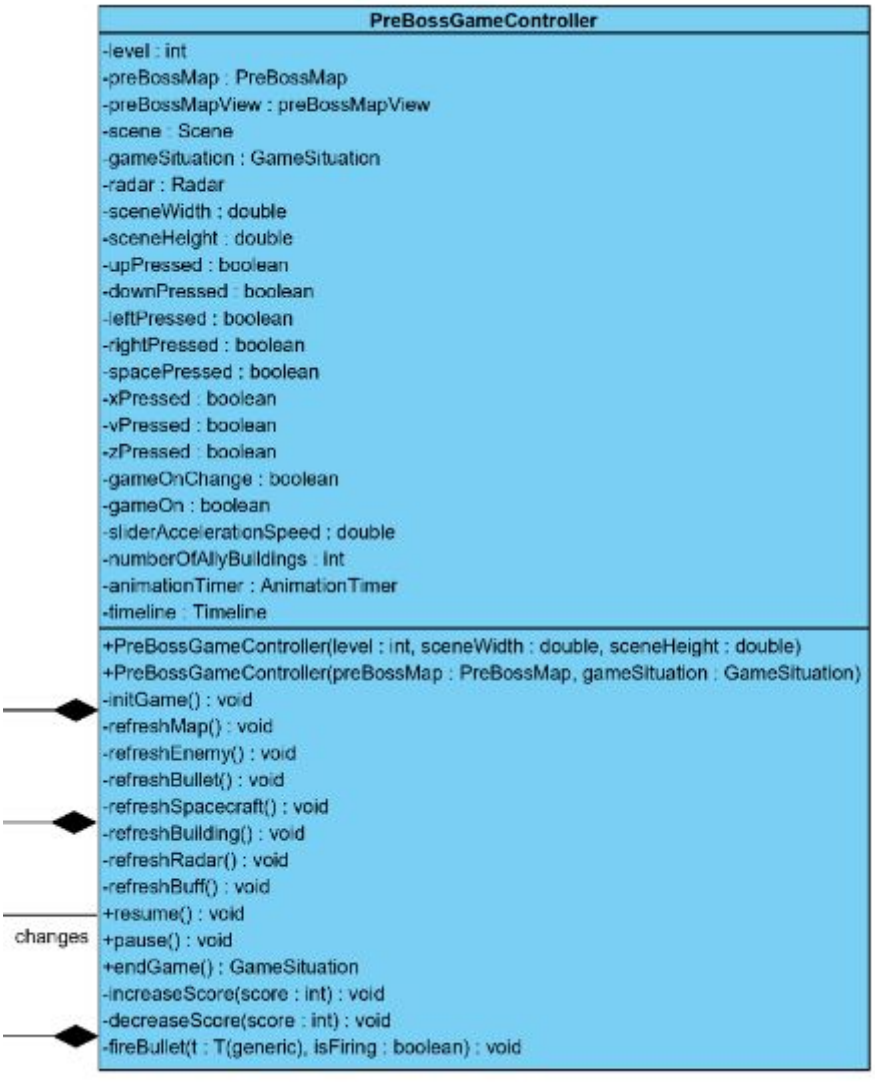
Part 2


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

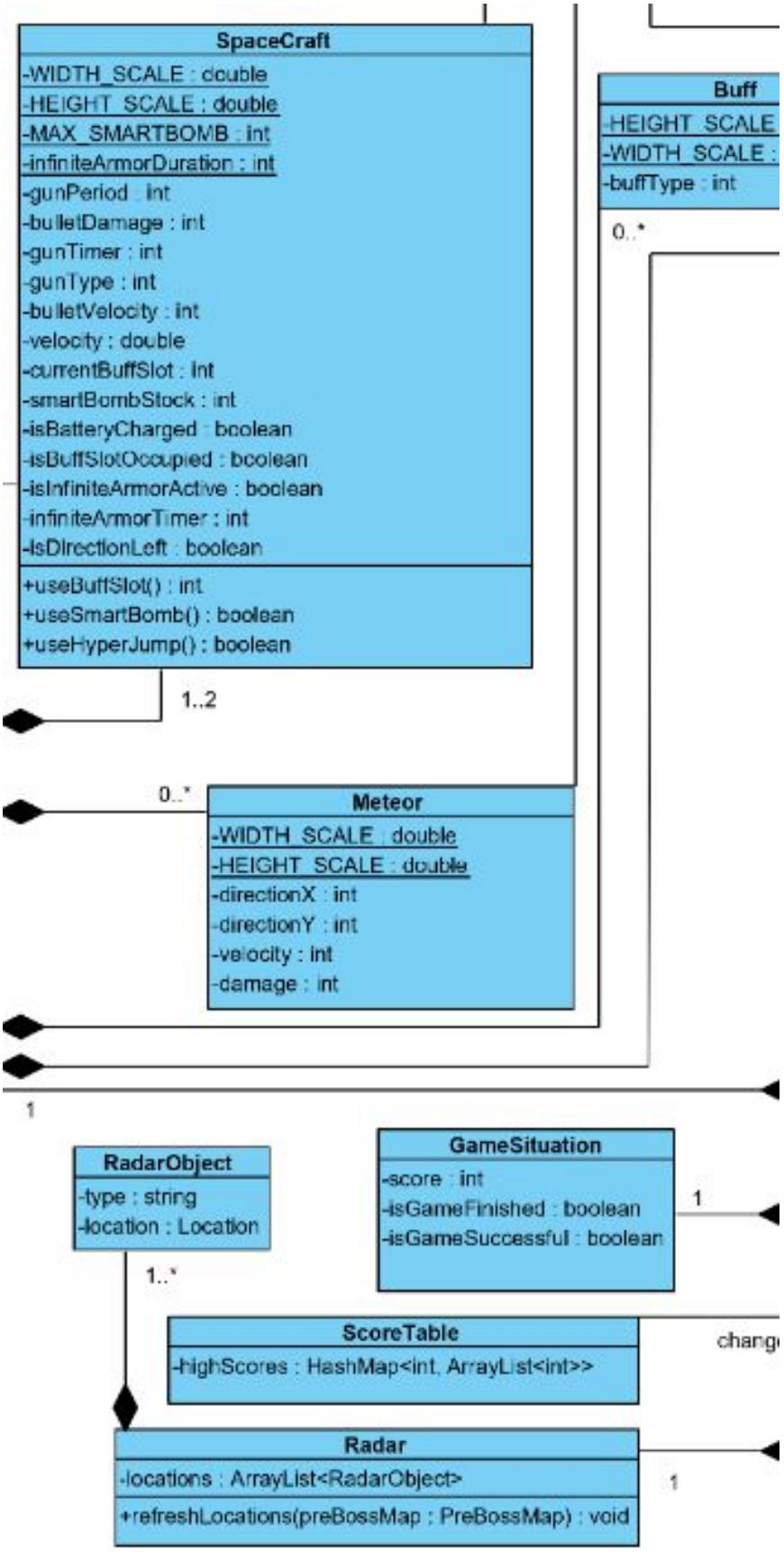
Part 3


<<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)
1
EnemyBuilding2
-hostingTime : int
-tier
-enemiesInside : HashMap<Long, Enemy>
-elapsedTimes : HashMap<Long, int>
-produceTimer : int
-producePeriod : int
+hostEnemy(enemy : Enemy) : void
+releaseEnemy(ID : long) : void
EnemyBuilding1
-tier : int
-produceTimer : int
-producePeriod : int
<<Interface>>
ProduceEnemy
+produceEnemy() : Enemy
+getProduceTimer() : int
+setProduceTimer(produceTimer : int) : void
+getProducePeriod() : int
+setProducePeriod(producePeriod : int) : void
SpaceCraft
-WIDTH_SCALE : double
-HEIGHT_SCALE : double
-MAX_SMARTBOMB : int
-infiniteArmorDuration : int
-gunPeriod : int
-bulletDamage : int
-gunTimer : int
-gunType : int
Buff
-HEIGHT_SCALE : double
-WIDTH_SCALE : double
-buffType : int
<<ORM Abstract Persistable>>
Building
-WIDTH_SCALE : double
-HEIGHT_SCALE : double
AllyBuilding
-isUnderAttack
0..*
0..*

Part 4


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
-refreshBuff() : void
+resume() : void
+pause() : void
+endGame() : GameSituation
-increaseScore(score : int) : void
-decreaseScore(score : int) : void
-fireBullet(t : T(generic), isFiring : boolean) : void
changes

Part 5


SpaceCraft
-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
-WIDTH_SCALE :
-buffType : int
0..*
1..2
0..*
Meteor
-WIDTH_SCALE : double
-HEIGHT_SCALE : double
-directionX : int
-directionY : int
-velocity : int
-damage : int
1
RadarObject
-type : string
-location : Location
1..*
GameSituation
-score : int
-isGameFinished : boolean
-isGameSuccessful : boolean
1
ScoreTable
-highScores : HashMap<int, ArrayList<int>>
Radar
-locations : ArrayList<RadarObject>
+refreshLocations(preBossMap : PreBossMap) : void
1

Explanation of All Classes

Location Class

Attributes:

- **private int positionX:** This attribute holds x value of the location.
- **private int positionY:** This attribute holds y value of the location

LocatableObject Class (Abstract)

Attributes:

- **private Location location:** This attribute holds location (x, y coordinates) of the object.
- **private double hitboxWidth:** This attribute holds hitbox width to determine the area which is covered by object.
- **private double hitboxHeight:** This attribute holds hitbox height to determine the area which is covered by object.
- **private long ID:** This attribute holds unique ID number of the object.
- **private static long currentID:** This attribute holds the static current ID to determine next object's ID.
- **private boolean isDead:** This attribute holds whether the object is dead in order to decide show it in the map.

- **private int healthPoint:** This attribute holds the current hp of the object. If this value is smaller than or equal to zero isDead becomes true.

Methods:

- **public void moveUp(double amount):** This method moves the object up a certain amount by changing the location.
- **public void moveDown(double amount):** This method moves the object down a certain amount by changing the location.
- **public void moveLeft(double amount):** This method moves the object left a certain amount by changing the location.
- **public void moveRight(double amount):** This method moves the object right a certain amount by changing the location.
- **public void moveToDirection(double velocity, double directionX, double directionY):** This method moves the object in the given directions. Movement amount is decided by the velocity.

Enemy Class (Abstract) inherits LocatableObject

Attributes:

- **private int damage:** This attribute holds the amount of damage inflicted by enemy.
- **private double velocity:** This attribute holds velocity of enemy object.

- **private int radarRadius:** This attribute holds the radius of enemy's field of vision.
- **private Location destinationLocation:** This attribute holds location of enemy's target place.
- **private String destinationType:** This attribute holds the type of target point such as spacecraft, alliance stations etc.
- **private int buffType:** This attribute holds buff type which will appear after the enemy is killed. It can be one of the several buff types or the enemy may not have any buff.
- **private boolean isEvolved:** This attribute holds whether the enemy is evolved or not.

Methods:

- **public Buff dropBuff():** This methods returns the contained buff if there is any when the Enemy dies.

Tier1 Class inherits Enemy, implements FireBullet

Attributes:

- **public static final WIDTH_SCALE:** This attribute holds the meaningful width scale value for 1080 pixel height. In order to achieve relative layout, this object's width needs to be equal to screens height x (WIDTH_SCALE/1080). In future, by changing this attribute from settings, user can have objects with

bigger widths. Therefore, this attribute is to ease possible changes to the game.

- **public static final HEIGHT_SCALE:** This attribute holds the meaningful height scale value for 1080 pixel height. In order to achieve relative layout, this object's height needs to be equal to screens height x (HEIGHT_SCALE/1080). In future, by changing this attribute from settings, user can have objects with bigger widths. Therefore, this attribute is to ease possible changes to the game.
- **private int bulletVelocity:** This attribute holds the velocity of Tier1's bullet.
- **private int bulletDamage:** This attribute holds the damage of Tier1's bullet.
- **private int gunPeriod:** This attribute holds the gunPeriod of Tier1's bullet meaning that Tier1 can fire bullets for every gunPeriod seconds.
- **private int gunTimer:** This attribute holds the gunTimer of Tier1's bullet. When gunTimer is equal to zero, a bullet can be shot. Then gunTimer starts to increase until it reaches to gunPeriod. When gunPeriod is reached gunTimer is reset to zero.

Tier2 Class inherits Enemy Class

Attributes:

- **public static final WIDTH_SCALE:** This attribute holds the meaningful width scale value for 1080 pixel height. In order to achieve relative layout, this object's width needs to be equal to screens height x (WIDTH_SCALE/1080).

In future, by changing this attribute from settings, user can have objects with bigger widths. Therefore, this attribute is to ease possible changes to the game.

- **public static final HEIGHT_SCALE:** This attribute holds the meaningful height scale value for 1080 pixel height. In order to achieve relative layout, this object's height needs to be equal to screens height x (HEIGHT_SCALE/1080). In future, by changing this attribute from settings, user can have objects with bigger widths. Therefore, this attribute is to ease possible changes to the game.
- **private int impact:** This attribute holds radius of Tier2's collapsing impact when Tier2 crashes to something.

Tier3 Class inherits Enemy, implements FireBullet

Attributes:

- **public static final WIDTH_SCALE:** This attribute holds the meaningful width scale value for 1080 pixel height. In order to achieve relative layout, this object's width needs to be equal to screens height x (WIDTH_SCALE/1080). In future, by changing this attribute from settings, user can have objects with bigger widths. Therefore, this attribute is to ease possible changes to the game.
- **public static final HEIGHT_SCALE:** This attribute holds the meaningful height scale value for 1080 pixel height. In order to achieve relative layout,

this object's height needs to be equal to screens height x (HEIGHT_SCALE/1080). In future, by changing this attribute from settings, user can have objects with bigger widths. Therefore, this attribute is to ease possible changes to the game.

- **private int bulletVelocity:** This attribute holds the velocity of Tier3's bullet.
- **private int bulletDamage:** This attribute holds the damage of Tier3's bullet.
- **private int gunPeriod:** This attribute holds the gunPeriod of Tier3's bullet meaning that Tier3 can fire bullets for every gunPeriod seconds.
- **private int gunTimer:** This attribute holds the gunTimer of Tier3's bullet. When gunTimer is equal to zero, a bullet can be shot. Then gunTimer starts to increase until it reaches to gunPeriod. When gunPeriod is reached gunTimer is reset to zero.

Methods:

- **public ArrayList<Enemy> divide():** This methods divides Tier3 enemy into Tier1 and Tier2 enemy and return these as a list to insert into map.

Bullet Class inherits LocatableObject

Attributes:

- **public static final WIDTH_SCALE:** This attribute holds the meaningful width scale value for 1080 pixel height. In order to achieve relative layout, this object's width needs to be equal to screens height x (WIDTH_SCALE/1080).

In future, by changing this attribute from settings, user can have objects with bigger widths. Therefore, this attribute is to ease possible changes to the game.

- **public static final HEIGHT_SCALE:** This attribute holds the meaningful height scale value for 1080 pixel height. In order to achieve relative layout, this object's height needs to be equal to screens height x (HEIGHT_SCALE/1080). In future, by changing this attribute from settings, user can have objects with bigger widths. Therefore, this attribute is to ease possible changes to the game.
- **private int damage:** This attribute holds the damage of the Bullet.
- **private int velocity:** This attribute holds the velocity of the Bullet.
- **private double directionX:** This attribute holds the X direction of the Bullet.
- **private double directionY:** This attribute holds the Y direction of the Bullet.
- The last three attribute are sent to moveToDirection method in LocatableObject class to arrange movement of the Bullet.

Spacecraft Class inherits LocatableObject

Attributes:

- **public static final WIDTH_SCALE:** This attribute holds the meaningful width scale value for 1080 pixel height. In order to achieve relative layout, this object's width needs to be equal to screens height x (WIDTH_SCALE/1080). In future, by changing this attribute from settings, user can have objects with

bigger widths. Therefore, this attribute is to ease possible changes to the game.

- **public static final HEIGHT_SCALE:** This attribute holds the meaningful height scale value for 1080 pixel height. In order to achieve relative layout, this object's height needs to be equal to screens height x (HEIGHT_SCALE/1080). In future, by changing this attribute from settings, user can have objects with bigger widths. Therefore, this attribute is to ease possible changes to the game.
- **public static final MAX_SMARTBOMB:** This attribute holds the boundaries of smart bombs.
- **public static infiniteArmorDuration:** This attribute holds the duration of infinite armor of the Spacecraft.
- **private int gunType:** This attribute holds the type of gun of the Spacecraft. For example simple gun fires one bullet and second type of gun fires two bullets.
- **private int bulletDamage:** This attribute holds the damage of the bullets of Spacecraft's gun.
- **private int bulletVelocity:** This attribute holds the velocity of Spacecraft's bullet.
- **private int gunPeriod:** This attribute holds the gunPeriod of Spacecraft's bullet meaning that Spacecraft can fire bullets for every gunPeriod seconds.
- **private int gunTimer:** This attribute holds the gunTimer of Spacecraft's bullet. When gunTimer is equal to zero, a bullet can be shot. Then gunTimer

starts to increase until it reaches to gunPeriod. When gunPeriod is reached gunTimer is reset to zero.

- **private double velocity:** This attribute holds the velocity of the Spacecraft.
- **private int currentBuffSlot:** This attribute holds the current buff in the buffslot of the Spacecraft.
- **private int smartBombStock:** This attribute holds the current numbers of the smart bombs in the Spacecraft's stock.
- **private boolean isBatteryCharged:** This attribute holds whether battery is charged to do a hyperjump.
- **private boolean isBuffSlotOccupied:** This attribute holds whether buff slot is occupied or not.
- **private boolean isInfiniteArmorActive:** This attribute holds the velocity of the Spacecraft.
- **private int infiniteArmorTimer:** This attribute holds the timer for the infinite armor of the Spacecraft. When it reaches to INFINITE_ARMOR_DURATION, infinite armor is deactivated.
- **private boolean isDirectionLeft:** This attribute holds the direction of the Spacecraft.

Methods:

- **public int useBuffSlot():** This method uses the buff in the slot and returns its value.

- **public boolean useSmartBomb():** This method tries to use a smart bomb. According to the stock, it returns whether smart bomb did used or couldn't be used.
- **public boolean useHyperJump():** This method tries to use a hyperjump. According to the battery, it returns whether hyperjump did used or couldn't be used.

FireBullet Interface

Methods:

- **public Bullet fireBullet():** This method is to fire bullets.

Other methods of this interface, which are getters and setters, are declared in this scope in order to use Java's polymorphism for classes that implements this interface.

Meteor Class inherits LocatableObject

Attributes:

- **public static final WIDTH_SCALE:** This attribute holds the meaningful width scale value for 1080 pixel height. In order to achieve relative layout, this object's width needs to be equal to screens height x (WIDTH_SCALE/1080). In future, by changing this attribute from settings, user can have objects with

bigger widths. Therefore, this attribute is to ease possible changes to the game.

- **public static final HEIGHT_SCALE:** This attribute holds the meaningful height scale value for 1080 pixel height. In order to achieve relative layout, this object's height needs to be equal to screens height x (HEIGHT_SCALE/1080). In future, by changing this attribute from settings, user can have objects with bigger widths. Therefore, this attribute is to ease possible changes to the game.
- **private int damage:** This attribute holds the damage of the Meteor.
- **private int velocity:** This attribute holds the velocity of the Meteor.
- **private double directionX:** This attribute holds the X direction of the Meteor.
- **private double directionY:** This attribute holds the Y direction of the Meteor.

The last three attributes are sent to moveToDirection method in LocatableObject class to arrange movement of the Meteor.

Buff Class inherits LocatableObject

Attributes:

- **public static final WIDTH_SCALE:** This attribute holds the meaningful width scale value for 1080 pixel height. In order to achieve relative layout, this object's width needs to be equal to screens height x (WIDTH_SCALE/1080). In future, by changing this attribute from settings, user can have objects with

bigger widths. Therefore, this attribute is to ease possible changes to the game.

- **public static final HEIGHT_SCALE:** This attribute holds the meaningful height scale value for 1080 pixel height. In order to achieve relative layout, this object's height needs to be equal to screens height x (HEIGHT_SCALE/1080). In future, by changing this attribute from settings, user can have objects with bigger widths. Therefore, this attribute is to ease possible changes to the game.
- **private int buffType:** This attribute holds the type of the Buff as integer. For example if it is 1, then it will behave as freeze buff.

Building Class (Abstract) inherits LocatableObject

Attributes:

- **public static final WIDTH_SCALE:** This attribute holds the meaningful width scale value for 1080 pixel height. In order to achieve relative layout, this object's width needs to be equal to screens height x (WIDTH_SCALE/1080). In future, by changing this attribute from settings, user can have objects with bigger widths. Therefore, this attribute is to ease possible changes to the game.
- **public static final HEIGHT_SCALE:** This attribute holds the meaningful height scale value for 1080 pixel height. In order to achieve relative layout, this object's height needs to be equal to screens height x

(HEIGHT_SCALE/1080). In future, by changing this attribute from settings, user can have objects with bigger widths. Therefore, this attribute is to ease possible changes to the game.

AllyBuilding inherits Building

Attributes:

- **private boolean isUnderAttack:** This attribute holds whether AllyBuilding is under attack or not. If it is under attack, a call for help will be seen at radar of the game.

EnemyBuilding1 inherits Building

Attributes:

- **private int tier:** This attribute holds the tier of the object to decide what kind of enemies this building can produce.
- **private int producePeriod:** This attribute holds the producePeriod of EnemyBuilding1 to produce enemy meaning that EnemyBuilding1 can produce an enemy for every producePeriod seconds.
- **private int produceTimer:** This attribute holds the produceTimer of EnemyBuilding1 to produce enemy. When produceTimer is equal to zero, an enemy can be produced, then produceTimer starts to increase until it reaches

to producePeriod. When producePeriod is reached produceTimer is reset to zero.

EnemyBuilding2 inherits Building

Attributes:

- **public int hostingTime:** This attribute holds the duration of evaluation process for this object.
- **private int tier:** This attribute holds the tier of the object to decide what kind of enemies this building can produce.
- **private int producePeriod:** This attribute holds the producePeriod of EnemyBuilding1 to produce enemy meaning that EnemyBuilding2 can produce an enemy for every producePeriod seconds.
- **private int produceTimer:** This attribute holds the produceTimer of EnemyBuilding2 to produce enemy. When produceTimer is equal to zero, an enemy can be produced, then produceTimer starts to increase until it reaches to producePeriod. When producePeriod is reached produceTimer is reset to zero.
- **private HashMap<Long, Enemy> enemiesInside:** This is the list of enemies currently evolving in this building.
- **private HashMap<Long, int> elapsedTime:** This is the timer list for enemies currently evolving in this building. When time of any enemy reaches to hostingTime, it will be released.

Methods:

- **public void hostEnemy(Enemy enemy):** This method adds given enemy into enemiesInside list for evolution process.
- **public enemy releaseEnemy(Long ID):** This method releases the enemy from the enemiesInside list as evolved.

ProduceEnemy Interface

Methods:

- **public Enemy produceEnemy():** This method is to produce enemies.

Other methods of this interface, which are getters and setters, are declared in this scope in order to use Java's polymorphism for classes implementing this interface.

PreBossMap Class implements (Java interface) Serializable

This class represents the current situation of the map. Where the buildings, enemies, or player's spacecraft are hold in as information in this class and constantly manipulated according to flow of the game. In order to provide data persistency, this class extends Serializable class from Java.

Attributes:

- **private double hitboxWidthScale:** This attribute holds the user's game screen width. This value is given by the PreBossGameController class in the constructor and changes when game screens width is changed. All of the hitboxes of objects on the map scales according to hitboxWidthScale and hitboxHeightScale.
- **private double hitboxHeightScale:** This attribute holds the user's game screen height. This value is given by the PreBossGameController class in the constructor and changes when game screens height is changed. All of the hitboxes of objects on the map scales according to hitboxWidthScale and hitboxHeightScale.
- **private double maxHeight:** This attribute holds maximum height of the map and scales according to hitboxHeightScale.
- **private double maxWidth:** This attribute holds maximum width of the map and scales according to hitboxWidthScale.
- **private int level:** This attribute keeps track of current level. This value is given by the PreBossGameController in the constructor and PreBossMap is designed according to the level.
- **private HashMap<Long, Enemy> enemies:** This attribute keeps track of the enemies the map contains.
- **private HashMap<Long, Building> buildings:** This attribute keeps track of the buildings in the map.

- **private HashMap<Long, Bullet> bullets:** This attribute keeps track of the bullets in the map.
- **private HashMap<Long, Buff> buffs:** This attribute keeps track of the buffs in the map.
- **private HashMap<Long, Meteor> meteors:** This attribute keeps track of the meteors in the map.
- **private ArrayList<Enemy> firingEnemies:** This attribute holds the enemies that is in a firing situation to a specific destination such as Spacecraft or AllyBuilding.
- **private ArrayList<Tier2> rushingEnemies:** This attribute holds the Tier2 enemies that are in a rushing situation to a specific destination such as Spacecraft or AllyBuilding.
- **private SpaceCraft spaceCraft:** This is an instance of SpaceCraft class, which represents space craft of player 1.
- **private SpaceCraft spaceCraft2:** This is an instance of SpaceCraft class, which represents space craft of player 2.

The slider is the area that is visible on the map for users.

- **private int viewLeft:** This value holds the sliders left coordinate according to the whole map.
- **private int viewRight:** This value holds the sliders right coordinate according to the whole map.
- **private Boolean isFrozen:** This value holds whether freezeGame buff is active or not. If it is active, objects other than Spacecraft will be frozen.

Methods:

- **public PreBossMap(int level, double hitboxWidthScale, double hitboxHeightScale):** This constructor initializes a map with a given level and size scales.
- **private void initMap():** This method creates starting objects on the map such as buildings or enemies according to the level.
- **public void checkMapSituation():** This method is to manipulate map according to its current situation. This method is called constantly except game is not in pause by PreBossGameController.
- **private <T extends LocatableObject> void checkCollision(LocatableObject obj, HashMap<Long, T>):** This method checks the given objects situation to other objects in the given list. For example any collision between two objects on the map is detected in here or radar check for enemies happens in here.
- **private void wanderAround(Enemy enemy):** This method deals with the movement of given enemies.
- **public void refreshMap():** This method updates class' maxHeight and maxWidth and calls refreshLocatableObject method for every list to scale them.
- **private <T extends LocatableObject> void refreshLocatableObject(HashMap<Long, T>, double widthScale, double heighScale):** When screen size's of user changes, this methods updates sizes and locations of specific list of LocatableObjects on the map according to their specific scale values. This method provides relative layout feature.

- **public void createEnemy(Enemy enemy):** This method adds a given enemy into the enemy list.
- **public void createBullet(Bullet bullet):** This method adds a given bullet into the bullet list.
- **public void createBuff(Buff buff):** This method adds a given buff after enemies die.
- **public void createMeteor(Meteor meteor):** This method adds a given meteor into the meteor list.
- **public void deleteBuilding(Long ID):** This method destroys building with a given ID.
- **public void deleteEnemy(Long ID):** This method destroys the enemy with given ID.
- **public void deleteBullet(Long ID):** This method destroys the bullet with given ID.
- **public void deleteBuff(Long ID):** This method destroys the buff with given ID.
- **public void deleteMeteor(Long ID) :** This method destroys the meteor with given ID.
- **public ArrayList<AllyBuilding> callHelp():** This method return's list of AllyBuildings that are under attack.
- **public void useSmartBomb():** This method uses the smart bomb according to the situation of Spacecraft and its stock.

- **public <T implements ProduceEnemy> void produceEnemy(ArrayList<T> list) :** This method handles production of enemies in EnemyBuilding1 and EnemyBuilding2.

RadarObject Class

Attributes:

- **private String type:** This attribute holds type of the RadarObject such as Spacecraft or Enemy or Building.
- **private Location location:** This attribute holds the location of the RadarObject.

Radar Class

Attributes:

- **private ArrayList<RadarObject> locations:** This attribute holds list of RadarObjects to be shown.

Methods:

- **public void refreshLocations(PreBossMap preBossMap):** This method updates the locations list according to given map.

GameSituation Class implements (Java interface) Serializable

In order to provide data persistency, this class extends Serializable class from Java.

Attributes:

- **private int score:** This attribute holds the current score of the map.
- **private boolean isGameFinished:** This attribute holds whether game is finished or not.
- **private boolean isGameSuccessful:** This attribute holds whether game reached to success situation or not.

ScoreTable Class implements (Java interface) Serializable

In order to provide data persistency, this class extends Serializable class from Java.

Attributes:

- **private HashMap<int, ArrayList<int>> highScores:** This attribute holds high scores for different levels.

PreBossGameController Class

Attributes:

- **private int level:** This attribute holds the current level of the game and it is given to this controller class in constructor. Rest of the attributes will be rooted according to the level.
- **private PreBossMap preBossMap:** This attribute holds the current map as model and will be used for business logic. This object will be serialized approximately every one minute in order to restore the game from it if there will be any error in the game to cause termination of the program.
- **private PreBossMapView preBossMapView:** This attribute is the view model of preBossMap model. It will change constantly according to the situation of the preBossMap.
- **private javafx.scene.Scene scene:** This attribute is to put javafx view nodes and draw them on to screen.
- **private GameSituation gameSituation:** This attribute holds the current situation of PreBossGameController object according to its score, success and continuation situation. This object will be serialized approximately every one minute in order to restore the game from it if there will be any error in the game to cause termination of the program.
- **private Radar radar:** This attribute holds the radar and changes constantly according to the situation of preBossMap.
- **private double sceneWidth:** This attribute holds the current width of the user's screen and changes if it changes. Rest of the objects such as preBossMap, preBossMapView, scene will scale according to this attribute.

- **private double sceneHeight:** This attribute holds the current height of the user's screen and changes if it changes. Rest of the objects such as preBossMap, preBossMapView, scene will scale according to this attribute.
- **private boolean upPressed:** This attribute holds whether up key is pressed in order to apply going up motion.
- **private boolean downPressed:** This attribute holds whether down key is pressed in order to apply going down motion.
- **private boolean leftPressed:** This attribute holds whether left key is pressed in order to apply going left motion.
- **private boolean rightPressed:** This attribute holds whether right key is pressed in order to apply going right motion.
- **private boolean spacePressed:** This attribute holds whether space key is pressed in order to apply fire bullet method for Spacecraft.
- **private boolean xPressed:** This attribute holds whether x key is pressed in order to apply current buff in the slot.
- **private boolean vPressed:** This attribute holds whether v key is pressed in order to apply hyperjump if battery is charged.
- **private boolean zPressed:** This attribute holds whether z key is pressed in order to apply smartbomb if there is any in the stock.
- **private boolean gameOnChange:** This attribute holds whether game on situation will change or not. If gameOnChange is true and gameOn is true, then gameOn will be false meaning that game will stop. Similarly, game will resume if gameOn is false at the beginning.

- **private boolean gameOn:** This attribute holds whether game is continuing or stopped.
- **private double sliderAccelerationSpeed:** This attribute is used to manipulate slider according to given input.
- **private int numberOfAllyBuildings:** This attribute helps to decide whether game ended successfully or not.
- **private javafx.animation.AnimationTimer animationTimer:** This attribute is used to create pulses for continuity of the game. The animationTimer will stop when the game needs to stop and will play if the game is on.
- **private javafx.animation.Timeline timeline:** This attribute is indefinitely playing timer to control all states of the game such as stop, resume, end.

Methods:

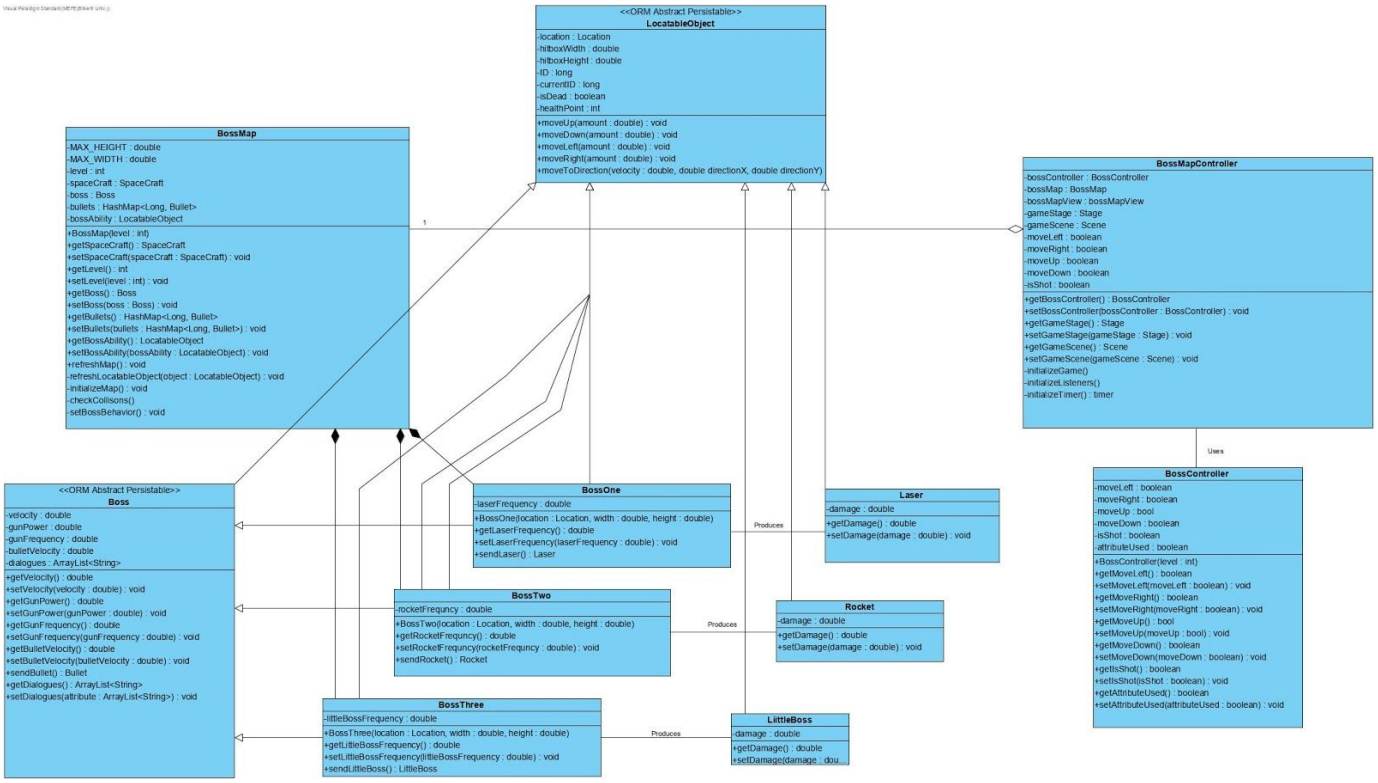
- **PreBossGameController(int level, double sceneWidth, double sceneHeight):** This is a constructor to build up the game from given level and size attributes.
- **PreBossGameController(PreBossMap preBossMap, GameSituation gameSituation):** This is a constructor to deserialize a game that has been crashed from its last saved situation.
- **private void initMap():** If given parameters in constructors are fine, this method will initialize the map according to them.
- **private void refreshMap():** This method is to refresh map at every pulse of animationTimer. Refreshing includes checking collisions, setting destinations for enemies, location changes in objects.

- **private void refreshEnemy():** This method is to reflect the changes in the preBossMap into view and model. For example informations that are taken from model will be given into view classes to show and if any object is dead then it will be deleted from the preBossMap.
- **private void refreshBullet():** This method is to reflect the changes in the preBossMap into view and model. For example informations that are taken from model will be given into view classes to show and if any object is dead then it will be deleted from the preBossMap.
- **private void refreshSpacecraft():** This method is to reflect the changes in the preBossMap into view and model. For example informations that are taken from model will be given into view classes to show and if any object is dead then it will be deleted from the preBossMap.
- **private void refreshMeteor():** This method is to reflect the changes in the preBossMap into view and model. For example informations that are taken from model will be given into view classes to show and if any object is dead then it will be deleted from the preBossMap.
- **private void refreshBuilding():** This method is to reflect the changes in the preBossMap into view and model. For example informations that are taken from model will be given into view classes to show and if any object is dead then it will be deleted from the preBossMap.
- **private void refreshBuff():** This method is to reflect the changes in the preBossMap into view and model. For example informations that are taken from model will be given into view classes to show and if any object is dead then it will be deleted from the preBossMap.

- **private void refreshRadar():** This method is to reflect the changes in the preBossMap into radar model and from there it will go to RadarView.
- **public void resume():** This method will resume the paused game.
- **public void pause():** This method will pause the resuming game.
- **private void increaseScore(int score):** This method will increase the current score with the given amount.
- **private void decreaseScore(int score):** This method will decrease the current score with the given amount.
- **private <T implements FireBullet> void fireBullet(T t, boolean isFiring):** This method is an helper method to generify firing bullet action to different classes.

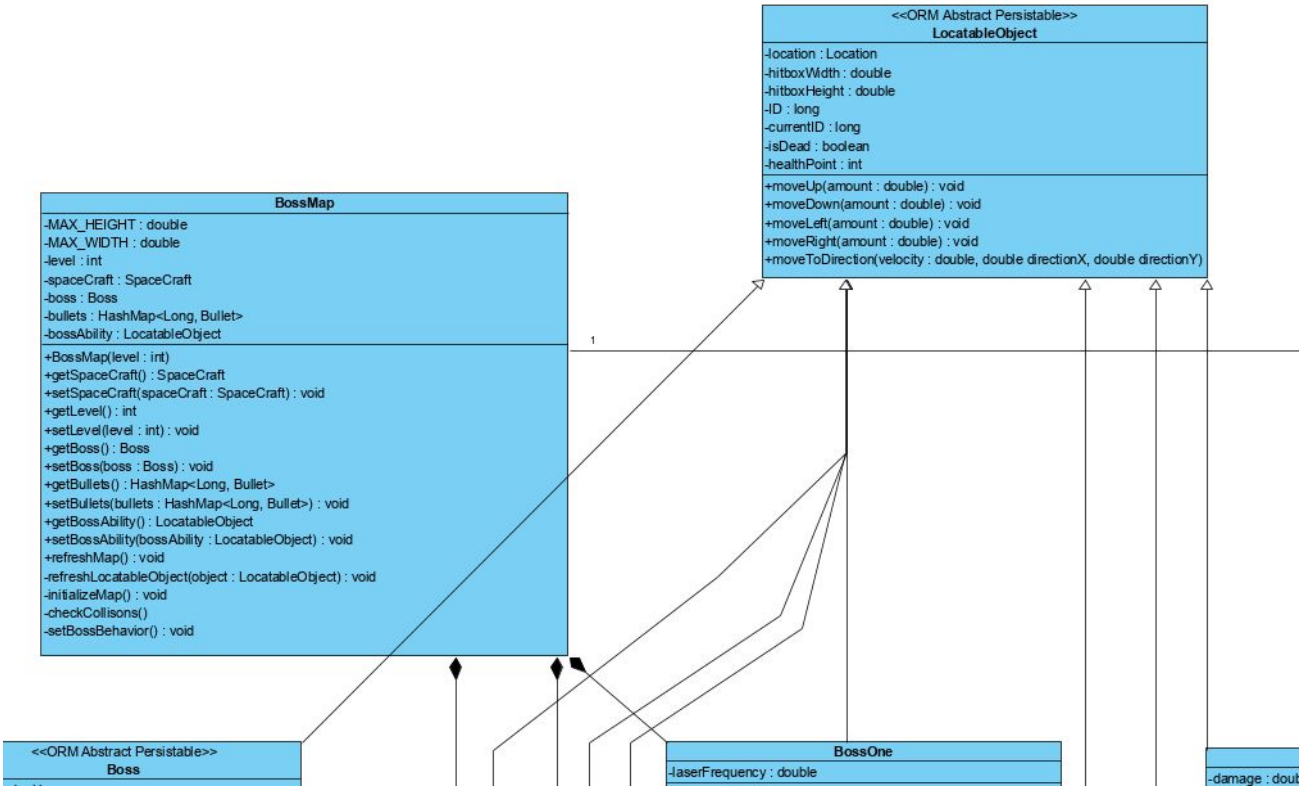
4.2. Boss Class Interfaces

Boss Controller-Model Classes All Parts

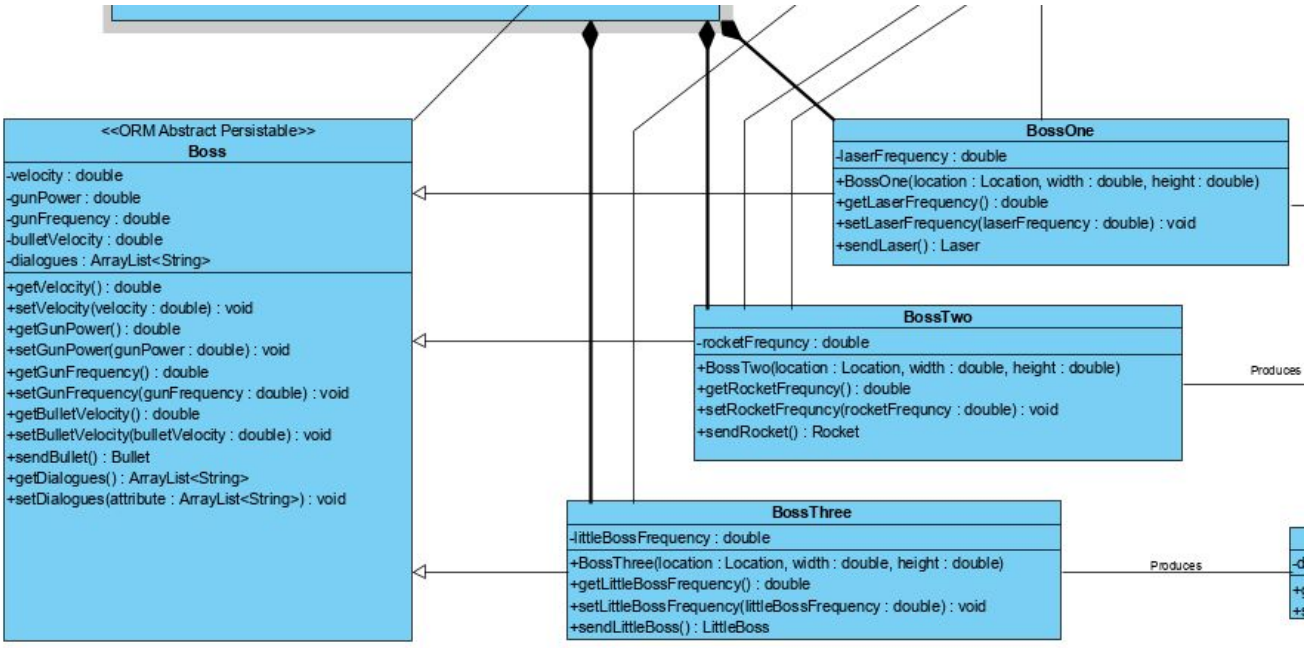


Boss Controller Model Classes Parts 1-5

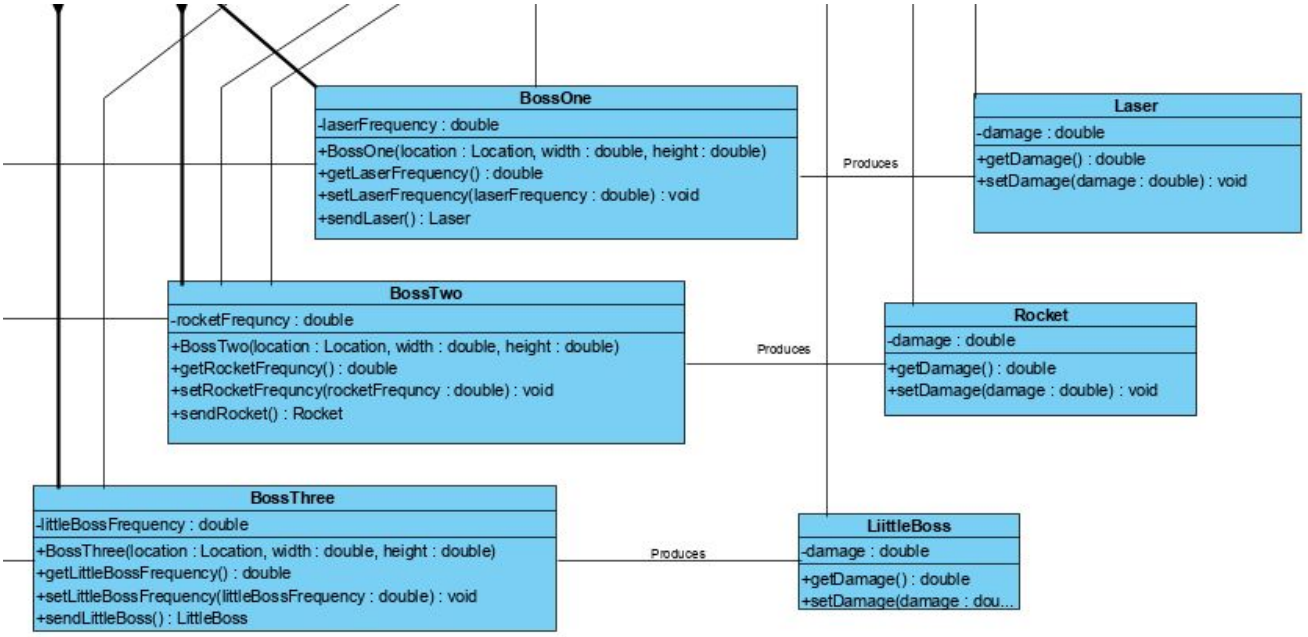
Part 1



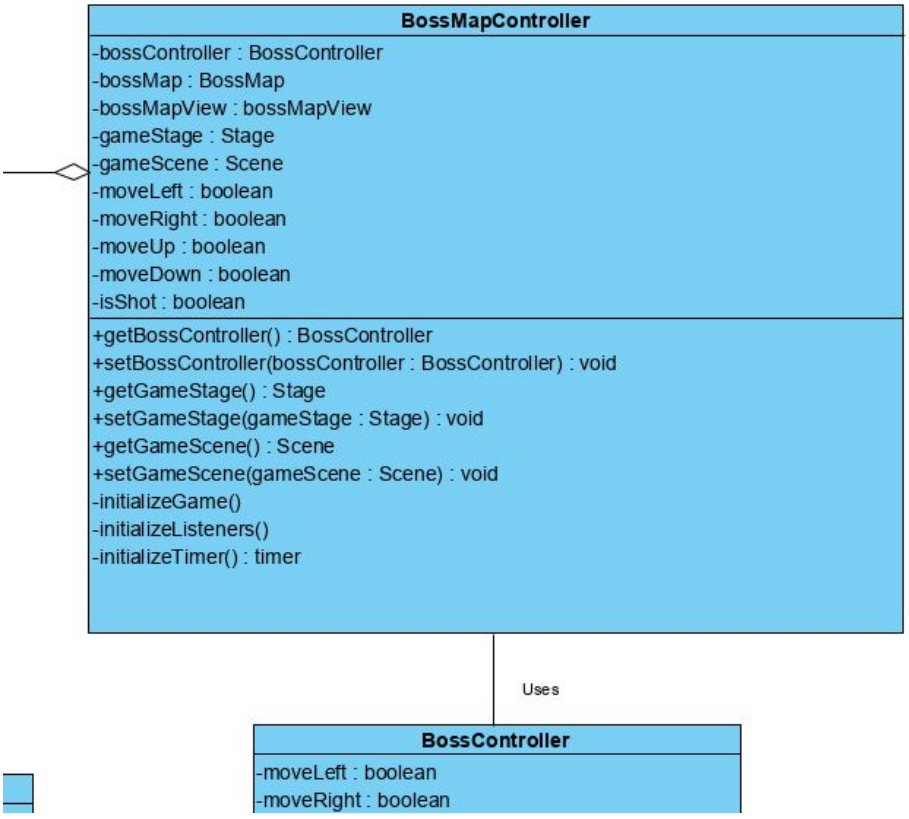
Part 2



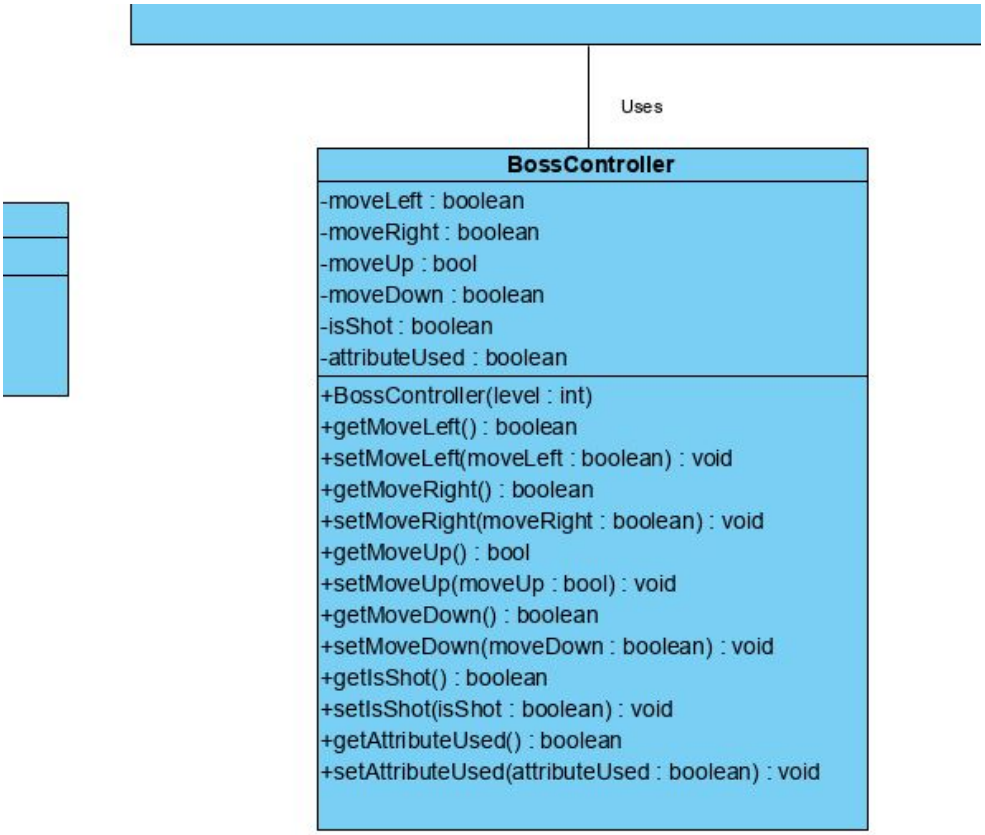
Part 3


BossOne
-laserFrequency : double
+BossOne(location : Location, width : double, height : double)
+getLaserFrequency() : double
+setLaserFrequency(laserFrequency : double) : void
+sendLaser() : Laser
Produces
Laser
-damage : double
+getDamage() : double
+setDamage(damage : double) : void
BossTwo
-rocketFrequncy : double
+BossTwo(location : Location, width : double, height : double)
+getRocketFrequncy() : double
+setRocketFrequncy(rocketFrequncy : double) : void
+sendRocket() : Rocket
Produces
Rocket
-damage : double
+getDamage() : double
+setDamage(damage : double) : void
BossThree
-littleBossFrequency : double
+BossThree(location : Location, width : double, height : double)
+getLittleBossFrequency() : double
+setLittleBossFrequency(littleBossFrequency : double) : void
+sendLittleBoss() : LittleBoss
Produces
LiittleBoss
-damage : double
+getDamage() : double
+setDamage(damage : dou...

Part 4


BossMapController
-bossController : BossController
-bossMap : BossMap
-bossMapView : bossMapView
-gameStage : Stage
-gameScene : Scene
-moveLeft : boolean
-moveRight : boolean
-moveUp : boolean
-moveDown : boolean
-isShot : boolean
+getBossController() : BossController
+setBossController(bossController : BossController) : void
+getGameStage() : Stage
+setGameStage(gameStage : Stage) : void
+getGameScene() : Scene
+setGameScene(gameScene : Scene) : void
-initializeGame()
-initializeListeners()
-initializeTimer() : timer
Uses
BossController
-moveLeft : boolean
-moveRight : boolean

Part 5



Explanation of Classes

LocatableObject Class

*** This class has been already explained in Pre-Boss Class Explanation.

Boss Abstract Class

Attributes:

- **private double velocity:** This attribute holds the velocity of Boss.

- **private double gunPower:** This attribute holds the power of Boss's gun.
- **private double gunFrequency:** This attribute holds the frequency of Boss's gun.
- **private double bulletVelocity:** This attribute holds the velocity Boss's bullets.
- **private ArrayList<String> dialogues:** This attribute holds the dialogues passing between Boss and the player.

Methods:

- **public Bullet sendBullet():** This method provides Boss to fire by sending bullet and it returns bullet.

BossOne Class

Attributes:

- **private double laserFrequency:** Boss Type 1 has laser as special power and this attribute holds the frequency of its laser.

Methods:

- **public BossOne(Location location, double width, double height):** This is constructor of Boss1 who takes location, width and height as parameter.
- **public Laser sendLaser():** This method provides Boss1 to use special power (laser) and it returns laser.

BossTwo Class

Attributes:

- **private double rocketFrequency:** Boss Type 2 has rockets as special power. It sends rockets to the points in the map which is determined by Boss. This attribute holds the frequency of sending rockets.

Methods:

- **public BossTwo(Location location, double width, double height):** This is constructor of Boss2 who takes location, width and height as parameter.
- **public Rocket sendRocket():** This method provides Boss2 to use special power (rocket) and it returns rocket.

BossThree Class

Attributes:

- **private double littleBossFrequency:** Boss Type 3 has Little Bosses as special power. This attribute holds the frequency of sending Little Bosses.

Methods:

- **public BossThree(Location location, double width, double height):** This is constructor of Boss3 who takes location, width and height as parameter.

- **public LittleBoss sendLittleBoss():** This method provides Boss3 to use special power (Little Boss) and it returns Little Boss.

Laser Class

Attributes:

- **private double damage:** This attribute holds the damage given by laser.

Rocket Class

Attributes:

- **private double damage:** This attribute holds the damage given by rocket.

LittleBoss Class

Attributes:

- **private double damage:** This attribute holds the damage given by Little Boss.

BossController Class

Attributes:

- **private boolean moveLeft:** This attribute holds whether Boss moves left to control Boss movements.
- **private boolean moveRight:** This attribute holds whether Boss moves right to control Boss movements.
- **private boolean moveUp:** This attribute holds whether Boss moves up to control Boss movements.
- **private boolean moveDown:** This attribute holds whether Boss moves down to control Boss movements.
- **private boolean isShot:** This attribute holds whether Boss fires bullet.
- **private boolean attributeUsed:** This attribute holds whether Boss uses special power.

Methods:

- **public BossController(int level):** This is constructor of BossController class which takes current level as parameter.

BossMap Class

Attributes:

- **private double MAX_HEIGHT:** This attribute holds maximum height of the map.

- **private double MAX_WIDTH:** This attribute holds maximum width of the map.
- **private int level:** This attribute keeps track of current level.
- **private SpaceCraft spaceCraft:** This is an instance of SpaceCraft class, which represents the player's spacecraft.
- **private Boss boss:** This is an instance of Boss class, which represents boss of this level.
- **private HashMap<Long, Bullet> bullets:** This attribute keeps track of the bullets in the map.
- **private LocatableObject bossAbility:** This is a locatable object which represents a special weapon for different types of bosses.

Methods:

- **public BossMap(int level):** This attribute keeps track of current level.
- **public void refreshMap():** This method organizes map again for changes. This method should be called continuously except pause.
- **public void refreshLocatableObject(LocatableObject object):** This method refreshes object in the game for changes. This method should be called continuously except pause.
- **public void initializeMap():** This method creates map according to level at the beginning of the Boss scene.
- **public void checkCollisons():** This methods controls collison of two objects.
- **public void setBossBehavior():** This methods arrange Boss's attackes and movements.

BossMapController Class

Attributes:

- **private BossController bossController:** This attribute is an instance of BossController class and enables BossMapController class to know Boss's actions.
- **private BossMap bossMap:** This attribute is an instance of BossMap class and enables BossMapController class to take map's current situation.
- **private BossMapView bossMapView:** This attribute is an instance of BossMapView class and enables BossMapController class to change map view.
- **private Stage gameStage:** This attribute is instance of Stage which represents game.
- **private Scene gameScene:** This attribute is instance of Scene which represents fighting with Boss scene.
- **private boolean moveLeft:** This attribute holds whether the player moves left according to pressed keys.
- **private boolean moveRight:** This attribute holds whether the player moves right according to pressed keys.
- **private boolean moveUp:** This attribute holds whether the player moves up according to pressed keys.
- **private boolean moveDown:** This attribute holds whether the player moves down according to pressed keys.

- **private boolean isShot:** This attribute holds whether the player shots according to pressed keys.

Methods:

- **public void initializeGame():** This methods arrange gameStage and gameScene
- **public void initializeListeners():** This method initialize JavaFX listeners.
- **public timer initializeTimer():** This method initializes timer and refresh map periodically.