Bilkent University

Department of Computer Engineering

CS319 Term Project

Defender: Fiyuv++

Design Report Iteration 2

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

Instructor: Eray TÜZÜN

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

Contents

1. Introduction **4**

1.1. Purpose of the System 4

1.2. Design Goals 4

1.2.1. Rapid Development 4

1.2.2. Portability 4

1.2.3. User-Friendliness 5

1.2.4. Design Trade-Offs 5

2. System Architecture **6**

2.1. Subsystem Decomposition 6

2.2. Hardware/Software Mapping 10

2.3. Persistent Data Management 11

2.4. Access Control Security 11

2.5. Boundary Conditions 12

2.5.1. Initialization 12

2.5.2. Termination 12

2.5.3. Failure 12

3. Subsystem Services **13**

3.1. Menu Management Subsystem 13

3.2. Pre Boss Management Subsystem 18

3.2.1. Pre Boss Model Management 19

3.2.2. Pre Boss View Management 27

3.3. Boss Management Subsystem 33

3.3.1. Boss Model Management 35

3.3.2. Boss View Management 40

3.4. File Management Subsystem 44

3.5. Asset Management Subsystem 45

4. Low Level Design 47

4.1. Final Object Design 47

4.1.1. Menu Class Interfaces 47

4.1.1.1. Interfaces of Controller Classes 48

4.1.1.2. Interfaces of View Classes 62

4.1.1.3. Interfaces of Model Classes 72

4.1.2. Pre Boss Scene Class Interfaces 78

4.1.2.1. Pre Boss Model Management Class Interfaces 78

4.1.2.2 Pre Boss View Management Class Interfaces 104

4.1.3. Boss Scene Class Interfaces 119

4.1.4. File Management Class Interfaces 138

4.1.5. Asset Management Class Interfaces 141

4.2. Design Decisions and Trade-offs 147

4.2.1. Pre Boss Game Design Decisions 147

4.2.2 Boss Game Design Decisions 148

4.2.3 Assets Manager Design Decisions 148

4.2.4 Menu Design Decisions 149

4.3. Packages 149

4.3.1. External Packages 149

4.3.2. Internal Packages 150

5. Improvement summary 151

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 teamwork 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. Rapid Development

We have time limit to develop our game and present it. Our total time was 3 months and we have 2 weeks left. Therefore, rapid development is one of the main goals of our game design. We tried to realistically design and report what we will be able to implement in this process.

1.2.2. Portability

We want our game to run in three different platforms. Therefore, we will test the game in Windows 10, Mac OS X 10.11 and Ubuntu 15.10. To facilitate the process of ensuring the game runs in mentioned platforms, we decided to use Java programming language. The language is known with the slogan “compile once, run everywhere.”.

1.2.3. User-Friendliness

While developing our game, user-friendliness was one of our design goals. In our game, 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 their purpose is. Also, the game will have a color option for people who are not comfortable with default color. A ‘How to Play’ page which is accessible from main page will also help players get used to the game environment. Also, the player will be able to pause the game using “ESC” key.

1.2.4. Design Trade-Offs

User-Friendliness vs Cost

The developers will spend much time to ensure that mentioned User-Friendliness criteria are met. It means we choose providing user friendly features even though it will take time of the developers.

Rapid Development vs Functionality

Because having a working game is more important than having more functionalities, we decided to drop some functionalities to catch up on time. For example, we were planning to implement ally buildings but we decided not to.

Usability vs Security

Although we could add authentication to the software which will provide different settings, high scores, game improvement to different users of one computer; we decided not to implement authentication. Main reason was to improve usability. By omitting the authentication stage, we made users to be able to directly focus on playing the game. However, we won’t have different user data because of our choice instead we will have different data for different computers.

2. System Architecture

2.1. Subsystem Decomposition

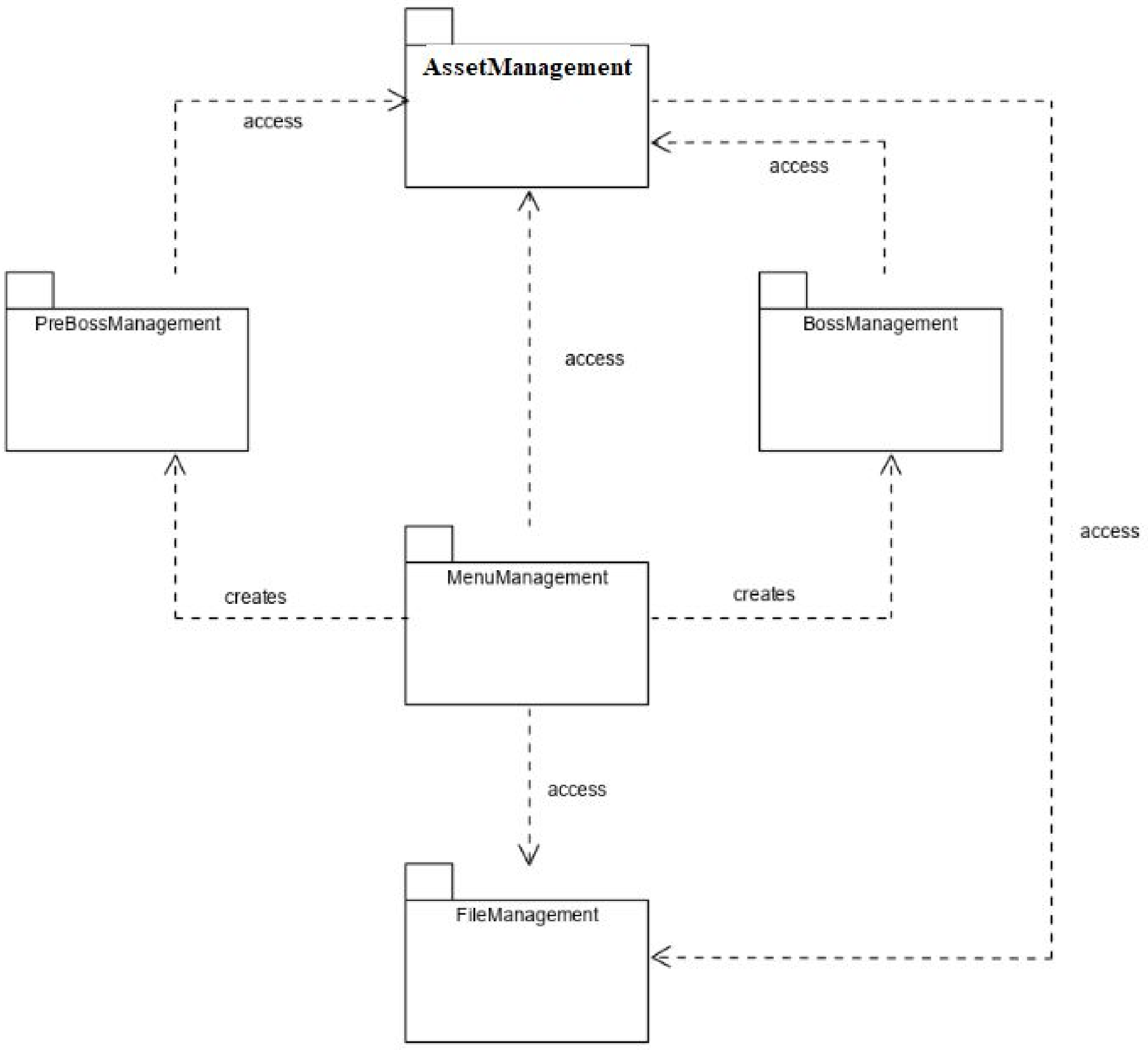
In this section, we will divide our project into subsystems. Because, it is difficult to understand the whole system without dividing it into pieces. Therefore, our purpose of subsystem decomposition is increasing the understandability and coherence of the system.

In the subsystem decomposition, we have decided to choose MVC (Model-View-Controller) structure as a design pattern for our project.

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 the separation of layers, it enables multiple people can work simultaneously on a project.

Diagram of the Subsystem Decomposition:



In our game, we have 5 different subsystems. These subsystems are Menu Management, File Management, Boss Management, PreBoss Management, and Asset Management.

Menu Management consists of classes that are related to the menu views and menu controllers. It briefly controls the menu transitions and initializes the game. Also, it makes the transition between PreBoss Management and Boss Management subsystems.

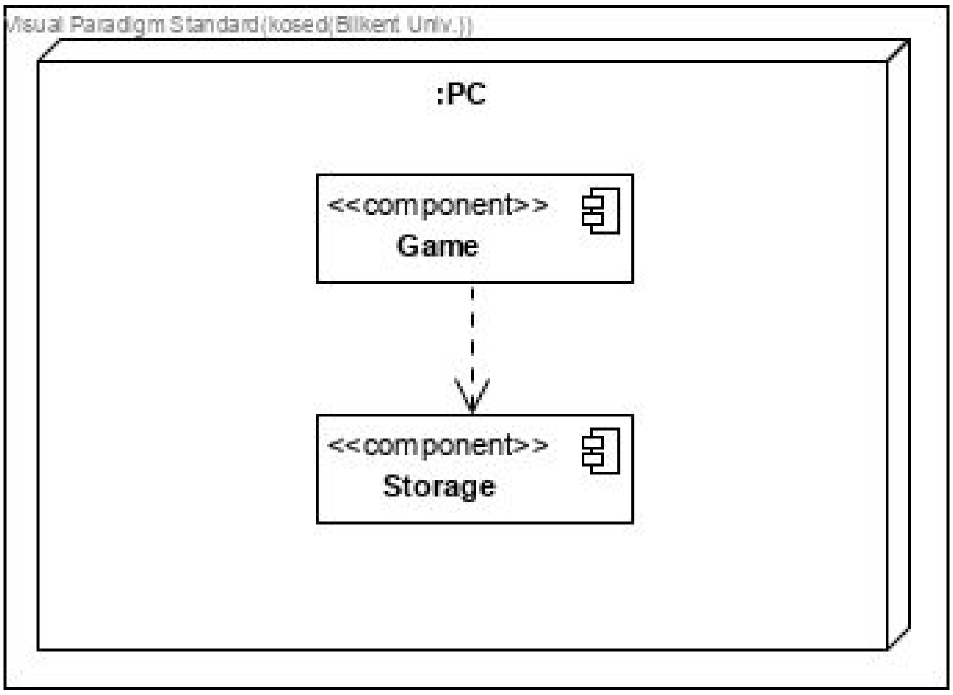
File Management provides the data from the text file to the Menu Management and Asset Management subsystems. Also, operations about saving and loading are handled in this subsystem.

PreBoss Management consists of classes that are related to the first scene of the game. It briefly controls the in-play game. Also, this subsystem loads related assets from Asset Management subsystem.

Boss Management is similar to the PreBoss Management subsystem because it consists of classes that are related to the second scene of the game and controls the second in-play scene of the game. Also, similar to the PreBoss Management, this subsystem accesses the Asset Management to get assets.

Asset Management is a kind of helper subsystem. It gets assets data in a raw form from the File Management subsystem at the beginning of the program and it converts the data into assets such as Image, Media, etc. Then, other subsystems can access these assets to initialize their view classes.

2.2. Hardware/Software Mapping



Since Fiyuv++ does not require any internet connection, the game has only one hardware node that is the user's personal computer. Therefore, the game and the save files will be stored in the computer that the game is deployed.

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 a 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.

2.3. Persistent Data Management

Fiyuv++ game has some specific data which needs to be persistent. For example, last preferred settings such as key bindings, level of volume, and theme need to be saved, because a user should not have to change these settings every time s/he restarts the game. Also, the current situation of the in-play game should be persistent in case there will be an error leading to a crash. Lastly, high scores of the game also need to be persistent, because when the game is restarted, high scores should not be deleted. 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 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 Fiyuv++ will not require any internet connection to play. Therefore, we have not login page or authentication and authorization of users for the game. Also, high scores and settings will be stored in a text file so that nobody can access the stored data in the game except the player itself. Due to the lack of internet connection, the 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 the text file. If the user changes settings from the menu, also these settings will be changed in the text file so that when the game is restarted, updated settings will be remained. Also, when the game initialized, all the assets will be uploaded from files.

2.5.2. Termination

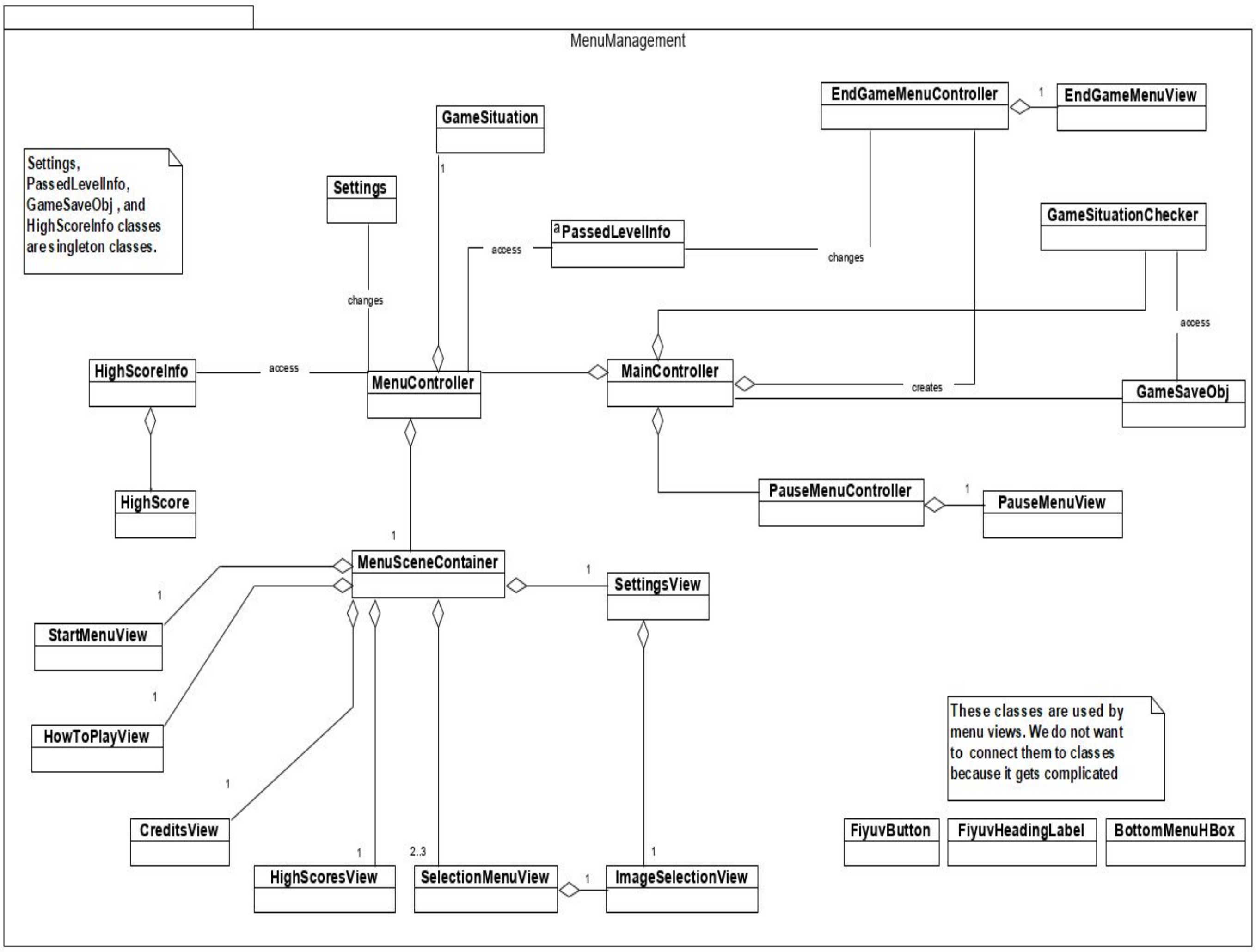
Termination of the game will be handled from different components. For instance, the game can be terminated from the main menu or the pause menu. When a user terminates the game, its settings and last process in the level will be saved into the text file.

2.5.3. Failure

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

3. Subsystem Services

3.1. Menu Management Subsystem



MainController Class:

An instance of this class will be initialized in main. This class will manage connection between FileManagement Subsystem and MenuManagement subsystem. When an information is needed to save, it uses an instance of

FileController and its methods for this process. It also deals with initializing necessary controllers when they are needed.

MenuController Class:

This class manages transition between menu screens before the game starts and make changes on single instances of GameInfo and SettingInfo class according to the player’s requests. It also informs MainController by changing BooleanProperty attributes about the player’s requests such as starting game or saving settings.

GameSituationChecker Class:

This class starts the game and manages transition from PreBoss section to Boss section. It informs MainController by changing BooleanProperty attributes when the game is paused and the game ends.

EndGameMenuController Class:

This class creates a pop up screen using an instance of EndGameMenuView class and informs MainControler by changing BooleanProperty attributes according to what player wants when the player presses buttons in End Game Menu.

PauseMenuController Class:

This class creates a pop up screen using an instance of PauseMenuView class and informs MainControler by changing BooleanProperty attributes according to what player wants when the player presses buttons in Pause Menu.

MenuSceneContainer Class:

This class contains instances of all Main Menu related view classes.

StartMenuView Class:

This class inherits AnchorPane from Javafx library and it contains buttons and layouts to design appearance of Start Menu.

SelectionMenuView Class:

This class inherits VBox from Javafx library and it contains layouts to design appearance of Spacecraft Selection Menu or Level Selection Menu. This class has also an instance of ImageSelectionView which creates a selection view with 3 images and 3 radio buttons with given sizes. This class also contains an instance of BottomMenu class.

ImageSelectionView Class:

This class inherits VBox from Javafx library and it creates a selection view with 3 images and 3 radio buttons with given sizes.

HighScoresView Class:

This class inherits BorderPane from Javafx library. This class contains tables and tabs to design appearance of HighScores Menu. It also contains an instance of BottomMenu.

SettingsView Class:

This class inherits VBox from Javafx library. This class contains layouts to design the appearance of Settings. It has text fields to get key selection from the user, an instance of ImageSelectionView class to get color selection from the user and an instance of bottom menu.

HowToPlayView Class:

This class inherits Pane from Javafx library. This class contains only a background image which shows how to play the game.

CreditsView Class inherits Pane:

This class inherits Pane from Javafx library. This class contains only a background image which shows information about the contributors of the game.

GameSaveObj Class:

This class implements Java Serializable interface. This class holds information that is needed to start last played game in order to let player resume.

GameInfo Class:

This class is singleton and it implements Java Serializable interface. This class holds information needed throughout the game. It also helps to restart last game by giving information about stages of the game and which spacecraft image is used.

SettingInfo Class:

This class is singleton and it implements Java Serializable interface. This class holds settings information. When the game is opened, saved settings will be uploaded.

PassedLevelInfo Class:

This class is singleton and it implements Java Serializable interface. It holds information about which levels are passed. This information is needed to know which levels should be locked.

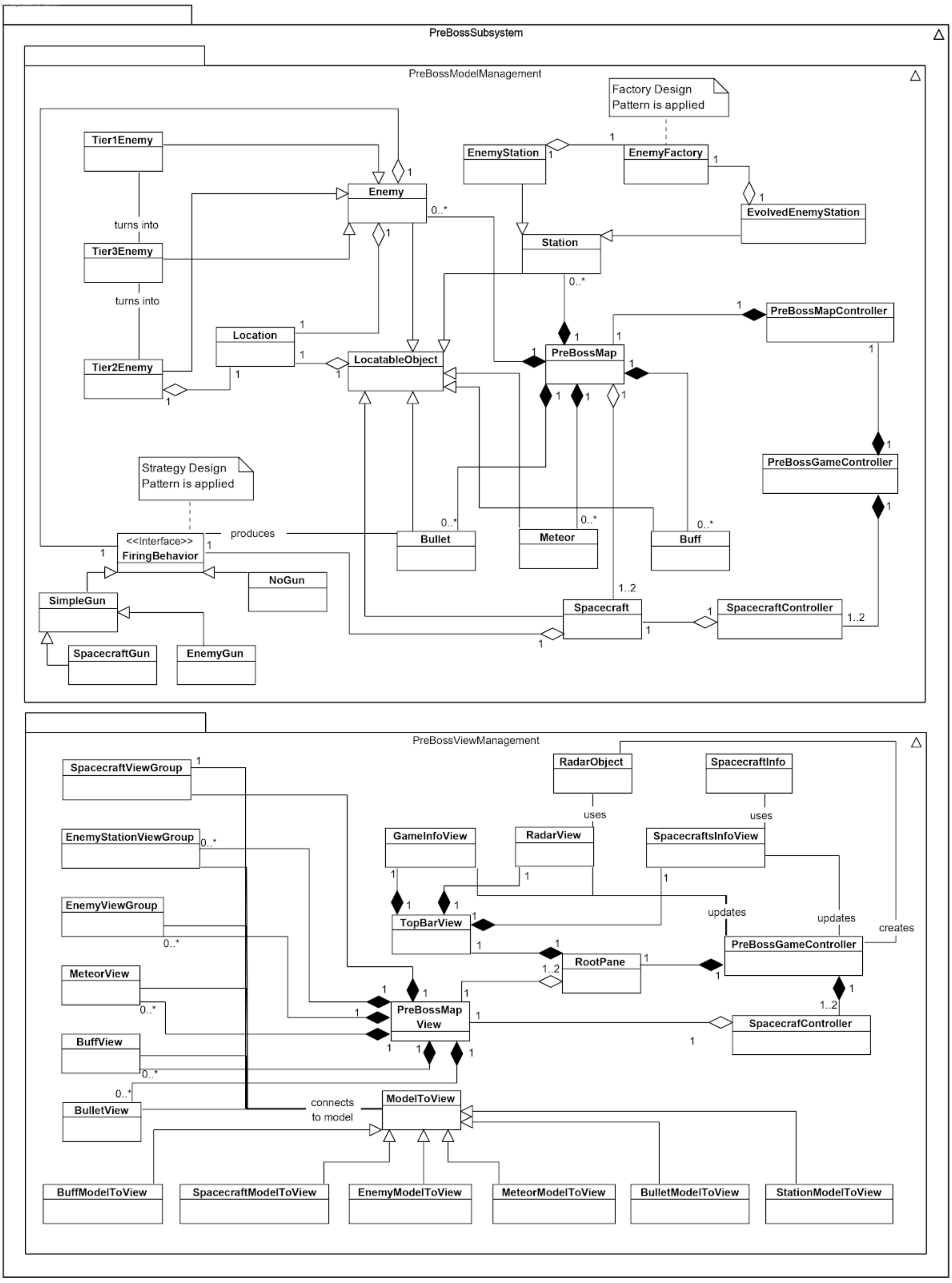
HighScoreInfo Class:

This class is singleton and it implements Java Serializable interface. This class holds high score information of Fiyuv++ games played in this computer. When the game is opened, high scores will be uploaded.

HighScore Class:

This class contains ranking and score information of a high score.

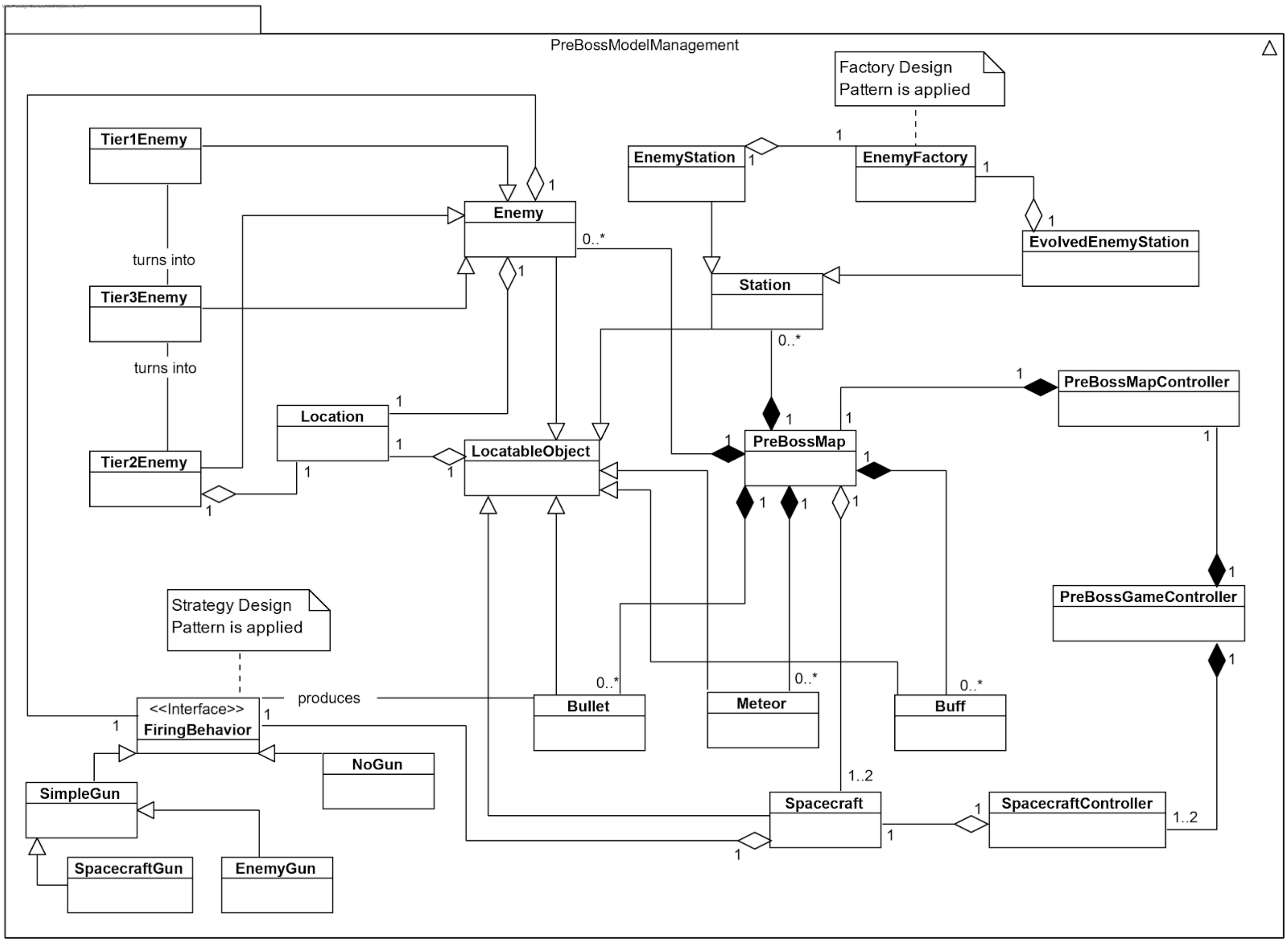
3.2. Pre Boss Management Subsystem



Pre Boss Scene Subsystem is responsible for the actions in the Pre Boss Scene which the spacecrafts fight with the simple enemies. In this subsystem, there are two parts which are Pre Boss Model Management and Pre Boss View Management parts.

3.2.1. Pre Boss Model Management

Pre Boss Model Management includes model classes of all game entities in the Pre Boss scene, pre boss game controller classes which tackle the connection between these Pre Boss scene model and Pre Boss scene view and handle business logic.



Location Class

This class represents a location on the map.

Enemy, LocatableObject, Tier2Enemy classes have one instances of Location Class; therefore, there are 1 to 1 aggregation relationships between these classes and Location class.

LocatableObject Class

This class is a superclass for any class that needs to be located on the map. It also provides uniqueness to objects on the map by giving them special IDs. This class is a abstract class, so only with the help of polymorphism there can be instances of this class.

Enemy Class

This class inherits LocatableObject class. It is a superclass for any enemy classes. This class provides fundamental information of an enemy. This class is a abstract class, so only with the help of polymorphism there can be instances of this class.

PreBossMap class can have multiple instances of this class and other than this there is no life-cycle for Enemy class; therefore, there is a 0..* to 1 composition relationship between this class and PreBossMap class.

Tier1Enemy Class

This class inherits Enemy class. It represents Tier 1 enemies on the map. This class provides specific information of a Tier 1 enemy.

Tier 3 enemies turn into Tier 1 enemies when they die.

Tier2Enemy Class

This class inherits Enemy class. It represents Tier 2 enemies on the map. This class provides specific information of a Tier 2 enemy.

Tier 3 enemies turn into Tier 2 enemies when they die.

Tier3Enemy Class

This class inherits Enemy class. It represents Tier 3 enemies on the map. This class provides specific information of a Tier 3 enemy.

Bullet Class

This class inherits LocatableObject class. It represents any bullet on the map and provides fundamental information of a bullet.

PreBossMap class can have multiple instances of this class and other than this there is no life-cycle for Bullet class; therefore, there is a 0..* to 1 composition relationship between this class and PreBossMap class. FiringBehavior interface produces new instances of this class.

Spacecraft Class

This class inherits LocatableObject class. It represents spacecrafts on the map and provides fundamental information of a spacecraft.

PreBossMap class can have 1 or 2 instances of this class and SpacecraftController has 1 instance of this class to control over it. There is a 1..2 to 1 aggregation relationship between this class and PreBossMap class and 1 to 1 aggregation relationship between this class and SpacecraftController class.

FiringBehavior Interface

This is a interface to provide strategy design pattern for firing bullet action.

Enemy and Spacecraft classes have one instances of FiringBehavior interface; therefore, there are 1 to 1 aggregation relationships between these classes and FiringBehavior interface.

SimpleGun Class

This class implements FiringBehavior interface. This class is a abstract class to represent common features of firing bullet actions.

EnemyGun Class

This class inherits SimpleGun class. This class holds necessary attributes of enemy firing bullet actions and provide a real bullet firing action for enemies.

SpacecraftGun Class

This class inherits SimpleGun class. This class holds necessary attributes of spacecraft firing bullet actions and provide a real bullet firing action for spacecrafts.

NoGun Class

This class implements FiringBehavior interface. This class represents the firing behavior of enemies without a gun.

Meteor Class

This class inherits LocatableObject class. It represents any meteor on the map and provides fundamental information of a meteor.

PreBossMap class can have multiple instances of this class and other than this there is no life-cycle for Meteor class; therefore, there is a 0..* to 1 composition relationship between this class and PreBossMap class.

<u>Buff Class</u>

This class inherits LocatableObject class. It represents any buff on the map and provides fundamental information of a buff.

PreBossMap class can have multiple instances of this class and other than this there is no life-cycle for Buff class; therefore, there is a 0..* to 1 composition relationship between this class and PreBossMap class.

<u>Station Class</u>

This class inherits LocatableObject class. It is a superclass for any enemy station classes. This class provides fundamental information of a station. This class is a abstract class, so only with the help of polymorphism there can be instances of this class.

PreBossMap class can have multiple instances of this class and other than this there is no life-cycle for Station class; therefore, there is a 0..* to 1 composition relationship between this class and PreBossMap class.

<u>EnemyStation Class</u>

This class inherits Station class. It represents simple enemy stations on the map and provides specific information a simple enemy station.

EvolvedEnemyStation Class

This class inherits Station class. It represents evolved enemy stations on the map and provides specific information an evolved enemy station.

EnemyFactory Class

This class is to provide Factory Design Pattern. Provides functions to produce enemy.

EnemyStation and EvolvedEnemyStation classes have one instances of EnemyFactory class; therefore, there are 1 to 1 aggregation relationships between these classes and EnemyFactory class.

PreBossMap Class

This class implements Java Serializable interface. It represents the map of Pre Boss Game stage. Essential entity of the pre boss game is this class. It includes and represents other entities.

PreBossMapController has 1 instance of this class to control over it. There is a 1 to 1 composition relationship between this class and PreBossMapController class.

PreBossMapController Class

This class provide a control over PreBossMap. Part of the business logic of the map happens in the functions of this class.

PreBossGameController has 1 instance of this class to give this class a life-cycle. There is a 1 to 1 composition relationship between this class and PreBossGameController class.

SpacecraftController Class

This class provide a control over a specific spacecraft. Business logic related to Spacecraft class happens in the functions of this class.

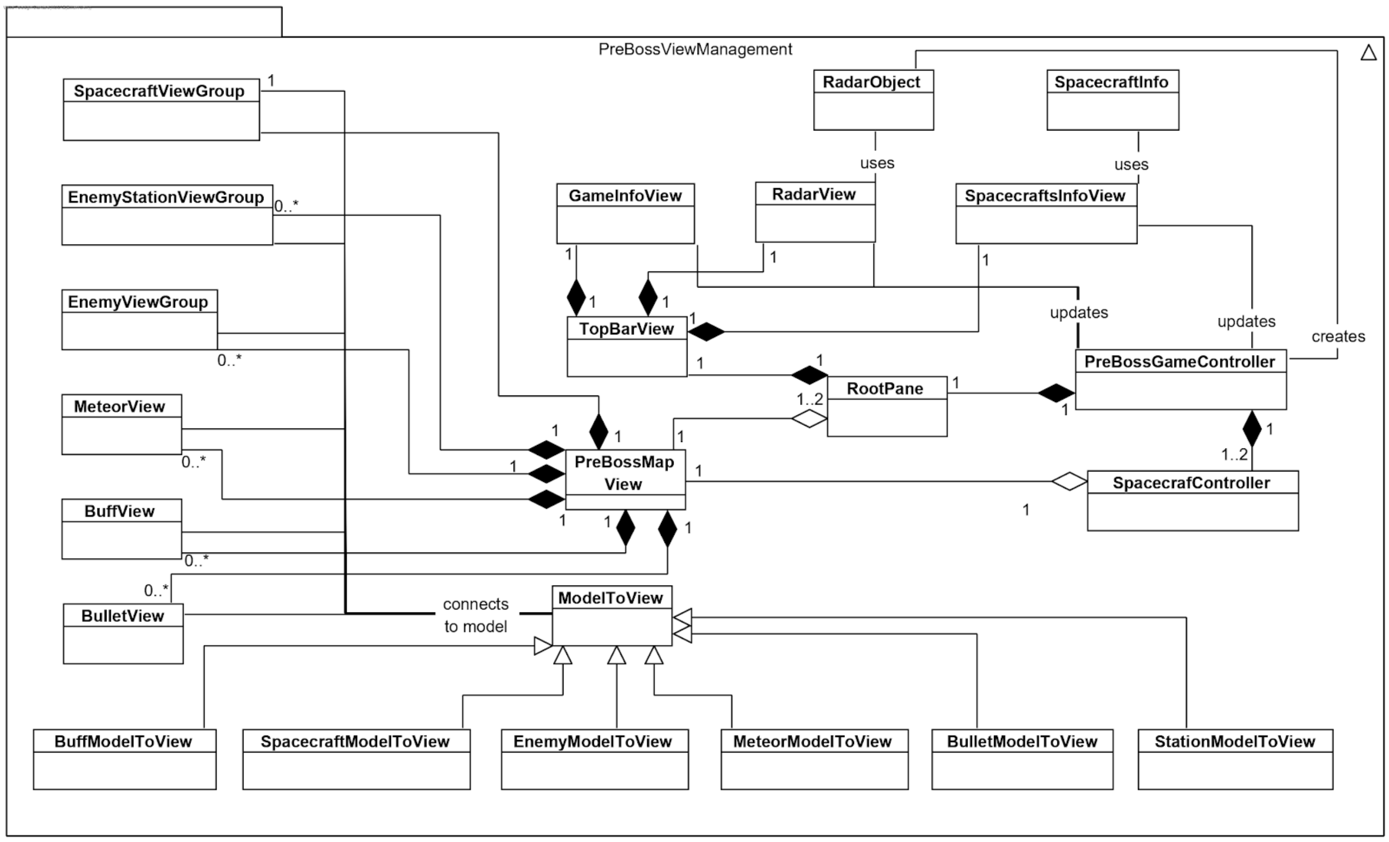
PreBossGameController has 1 or 2 instances of this class to give this class a life cycle. There is a 1..2 to 1 composition relationship between this class and PreBossGameController class.

PreBossGameController Class

This class provides the general business logic of the Pre Boss Game stage. This class connects the user input to the Pre Boss Game stage and provides continuity to the game.

3.2.2. Pre Boss View Management

In Pre Boss View Management, there are view classes which holds views of Pre Boss Scene entities, Pre Boss Map View class to refresh the position of every entity, and controller classes to make connection between view and model classes.



SpacecraftViewGroup Class

EnemyStationViewGroup Class

EnemyViewGroup Class

MeteorView Class

BuffView Class

BulletView Class

The classes above are to provide screen view related to their scope. Their life cycles are in PreBossMapView class; therefore, there are composition relationships between these classes and PreBossMapView.

The connection between these view classes and respecting model classes are built by ModelToView class.

ModelToView Class

This class provides the most general information necessary to put a model as a view on the screen.

BuffModelToView Class

SpacecraftModelToView Class

<u>EnemyModelToView Class</u>

<u>MeteorModelToView Class</u>

<u>BulletModelToView Class</u>

<u>StationModelToView Class</u>

The classes above are to provide more detailed information from model to view representation of entities specific to their scope. These classes inherit ModelToView class to provide superficial information for view.

<u>PreBossMapView Class</u>

This class represents the screen view of the map of Pre Boss Game stage. Essential entity of the pre boss game view is this class. It includes and represents other view entities.

SpacecraftController has 1 instance of this class to control over it for spacecraft related business logic. Since a spacecraft is positioned at the center of a map view, if there is two spacecraft then there should be two PreBossMapView. There is a 1 to 1 composition relationship between this class and PreBossMapView class. Similarly RootPane has 1 or 2 instances of this class (1 if single player, 2 if not) to control over it other than spacecraft related business logic. There is a 1..2 to 1 composition relationship between this class and RootPane class.

GameInfoView Class

This class represents the screen view of information about game’s current situation such as how many enemies left or what is score.

TopBarView class has 1 instance of this class; therefore, there are 1 to 1 composition relationships between this class and TopBarView interface.

PreBossGameController class updates this view class.

RadarView Class

This class represents the screen view of current positions of the objects on the map as a radar. In other words, this class is the bird's eye view version of Pre Boss Map View class.

TopBarView class has 1 instance of this class; therefore, there are 1 to 1 composition relationships between this class and TopBarView interface.

PreBossGameController class updates this view class.

RadarObject Class

This class represents the necessary information for a object to be part of the radar view.

RadarView class uses this class to connect entity models to itself.

PreBossGameController creates instances of this class and sends as a parameter to RadarView according to the map model.

SpacecraftInfoView Class

This class represents the screen view of information related to spacecrafts such as their smart bomb stock, health points.

TopBarView class has 1 instance of this class; therefore, there are 1 to 1 composition relationships between this class and TopBarView class.

PreBossGameController class updates this view class.

SpacecraftInfo Class

This class represents the necessary information for a spacecraft to be shown at SpacecraftInfoView class.

SpacecraftInfoView class uses this class to connect spacecraft entity models to itself.

PreBossGameController creates instances of this class and sends as a parameter to RadarView according to the map model.

TopBarView Class

This class holds SpacecraftInfoView, RadarView, GameInfoView together to handle their layouts.

RootPane class has 1 instance of this class; therefore, there are 1 to 1 composition relationships between this class and RootPane class.

RootPane Class

This class holds TopBarView, PreBossMapView together to handle their layouts.

PreBossGameController class has 1 instance of this class; therefore, there are 1 to 1 composition relationships between this class and PreBossGameController class.

PreBossGameController Class

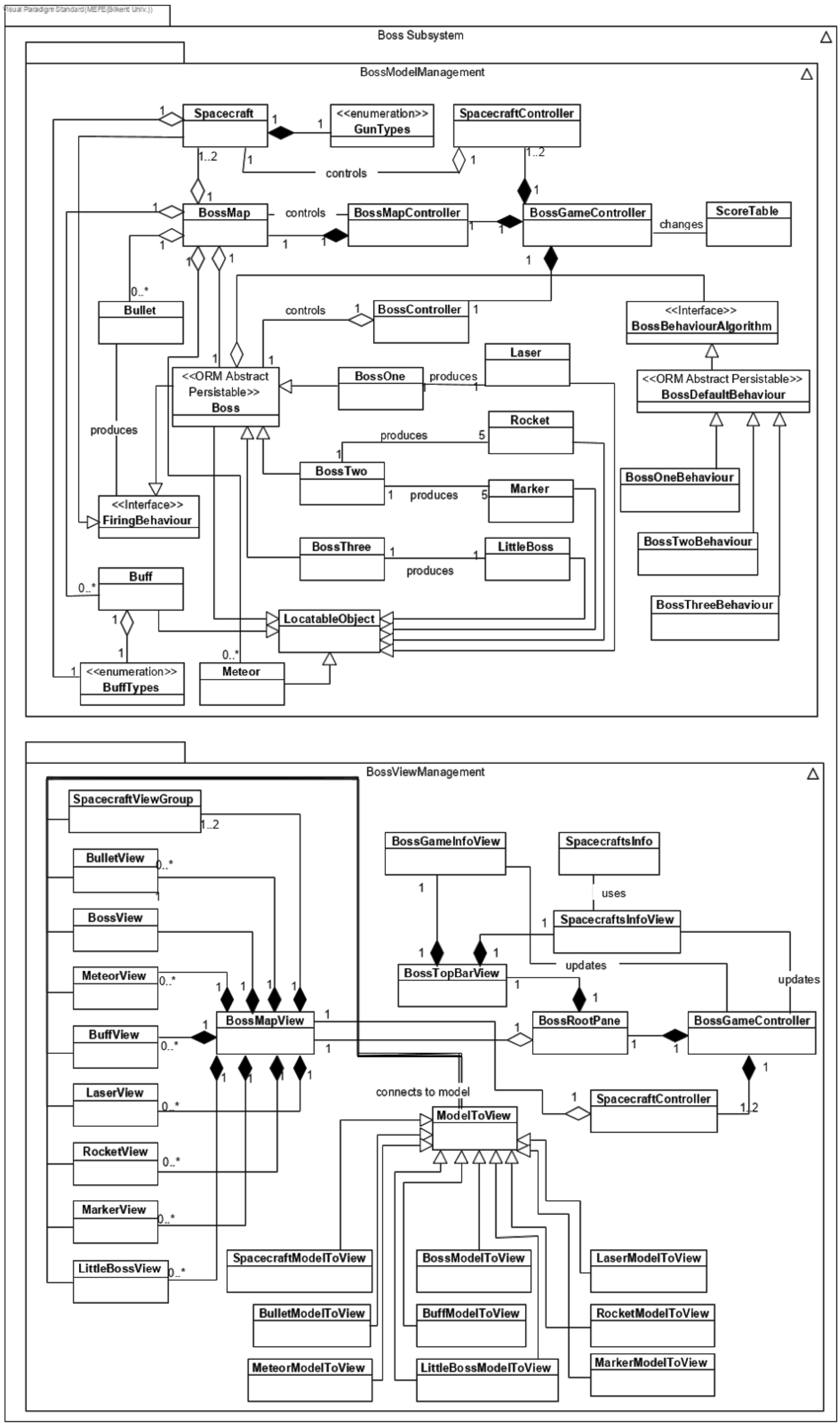
This class refreshes map related view classes according to their model entities.

SpacecraftController Class

This class refreshes spacecraft related view classes according to their model entities.

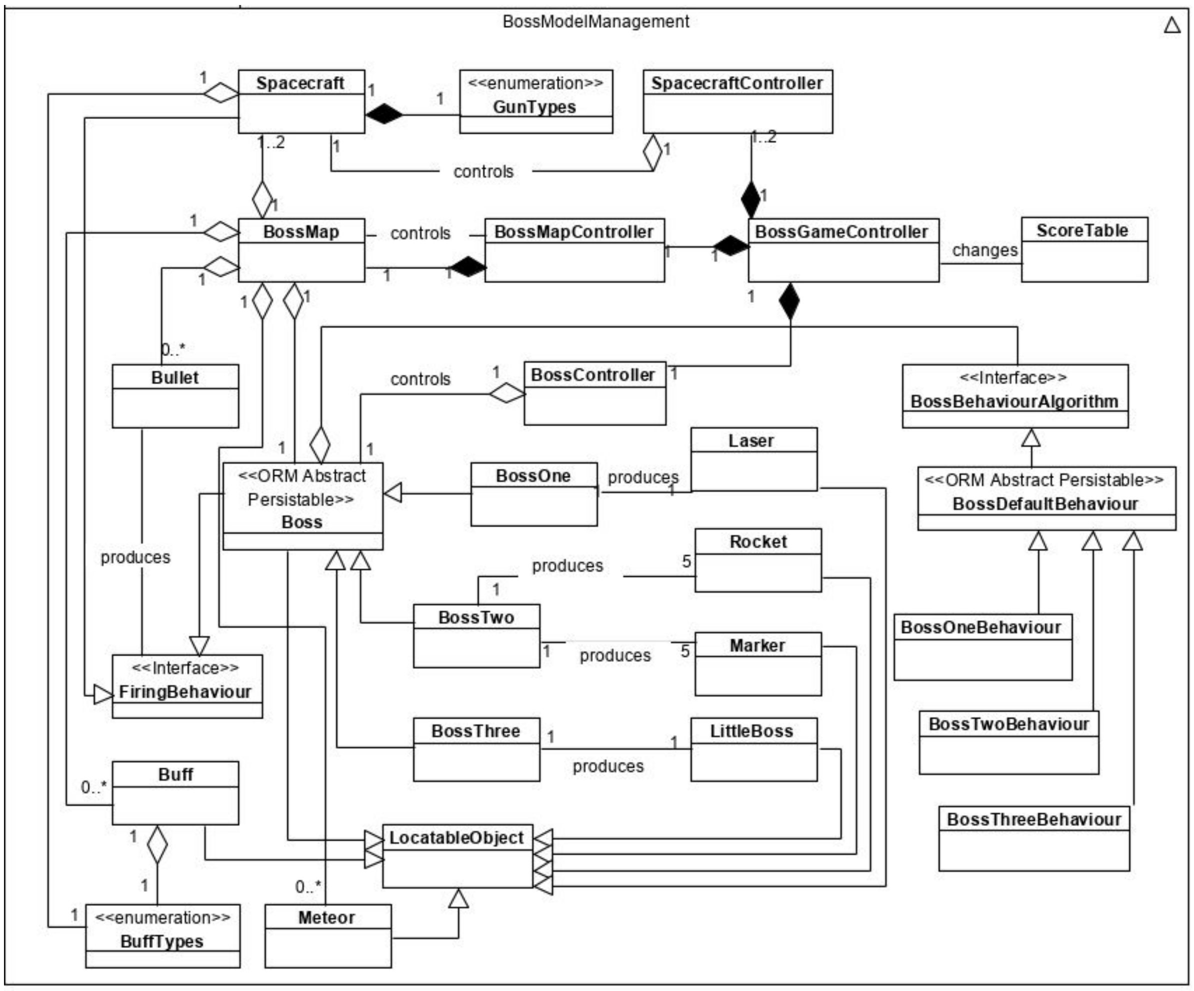
3.3. Boss Management Subsystem

Boss Scene Subsystem are responsible for the actions in the Boss Scene which the spacecrafts fight with the Boss. In this subsystem, there are two parts which are Boss Model Management and Boss View Management parts.


Visual Paradigm Standard(MEFE(Bilkent Univ.))
Boss Subsystem
BossModelManagement
Spacecraft
<<enumeration>> GunTypes
SpacecraftController
controls
BossMap
controls
BossMapController
BossGameController
changes
ScoreTable
Bullet
controls
BossController
<<Interface>> BossBehaviourAlgorithm
Laser
<<ORM Abstract Persistable>> Boss
BossOne
produces
<<ORM Abstract Persistable>> BossDefaultBehaviour
Rocket
produces
produces
BossTwo
Marker
produces
<<Interface>> FiringBehaviour
BossOneBehaviour
BossTwoBehaviour
BossThree
LittleBoss
produces
Buff
BossThreeBehaviour
LocatableObject
<<enumeration>> BuffTypes
Meteor
BossViewManagement
SpacecraftViewGroup
BulletView
BossView
MeteorView
BuffView
LaserView
RocketView
MarkerView
LittleBossView
BossGameInfoView
SpacecraftsInfo
uses
SpacecraftsInfoView
BossTopBarView
updates
updates
BossMapView
BossRootPane
BossGameController
connects to model
SpacecraftController
ModelToView
SpacecraftModelToView
BossModelToView
LaserModelToView
BulletModelToView
BuffModelToView
RocketModelToView
MeteorModelToView
LittleBossModelToView
MarkerModelToView

3.3.1. Boss Model Management

Boss Model Management includes model classes of all game entities in the Boss scene, boss game controller class which tackles the connection between these Boss scene model and Boss scene view.



CLASSES IN BOSS MODEL MANAGEMENT

Boss Abstract Class

This class is a simple abstract class to represent common features of different types of bosses. The class extends LocatableObject Class to use its functionalities and implements FiringBehaviour interface to be able to fire bullet. It also implements BossBehaviorAlgorithm to decide movements of the Boss.

BossBehaviourAlgorithm Interface

This interface provides necessary methods for bosses' behaviours such as shooting, moving, or using special ability. If there is a need for a new Boss, one can add the behaviour of the new Boss here. The interface is created as a result of strategy design pattern. Given methods can change from one Boss to another.

BossDefaultBehavior Abstract Class

The class contains the default behaviours of the Boss. It implements BossBehaviourAlgortihm.

<u>BossOneBehavior Class</u>

BossOne behaves according to this class. The class extends BossDefaultBehaviour to get predetermined -behaviours that do not change from one boss to another- behaviours.

<u>BossTwoBehaviour Class</u>

BossTwo behaves according to this class.

<u>BossThreeBehaviour Class</u>

BossThree behaves according to this class.

**** Because of the strategy design pattern, the methods of the BossTwoBehaviour and BossThreeBehaviour classes have the same functionality with BossOneBehaviour. The only difference is implementation.*

<u>BossOne Class</u>

This class represents Boss of the first level. The class extends Boss Class to use its functionalities. It also includes another functionalities related to special ability of Boss1 which is laser.

BossTwo Class

This class represents Boss of the second level. The class extends Boss Class to use its functionalities. It also includes another functionalities related to special ability of Boss2 which is rocket.

BossThree Class

This class represents Boss of the third level. The class extends Boss Class to use its functionalities. It also includes another functionalities related to special ability of BossThree which is little boss.

Laser Class

This class represent laser object. It is produced by BossOne. Since BossOne can produce 1 laser at a moment, there is 1 to 1 association relationship between them.

Rocket Class

This class represent rocket object. It is produced by BossTwo. Since BossTwo can produce 5 rockets at a moment, there is 1 to 5 association relationship between them.

Marker Class

This class represent marker which is produced to show where the rockets are thrown. Therefore, it is produced by BossTwo as well. Similarly to rockets, BossTwo can produce 5 markers at a moment and there is 1 to 5 association relationship between them.

LittleBoss Class

This class represent little boss object. It is produced by BossThree. Since BossThree can produce 1 little boss at a moment, there is 1 to 1 association relationship between them.

BossController Class

This class controls the actions of the boss. It is also a part of BossGameController. Therefore, there is 1 to 1 composition relationship between these two controller classes.

BossMap Class

This class represents all game entities in the Boss Scene. Therefore, it has a aggregation relationship with game entities.

BossMapController Class

This controller class provides the connection between BossMap and BossMapView. It checks game entities' situations and checks if there is any collision between game entities. In this way, every change in the map is transmitted to view classes.

<u>BossGameController Class</u>

This class is responsible for overall game flow. It includes BossController, SpacecraftController, and BossMapController to manage them together. Therefore, it has 1 to 1 composition relationships with BossController and BossMapController. However, there is 1..2 to 1 composition between SpacecraftController and BossGameController since Fiyuv++ has also two-players game mode. In two-players game mode, there has to be 2 controllers for spacecrafts.

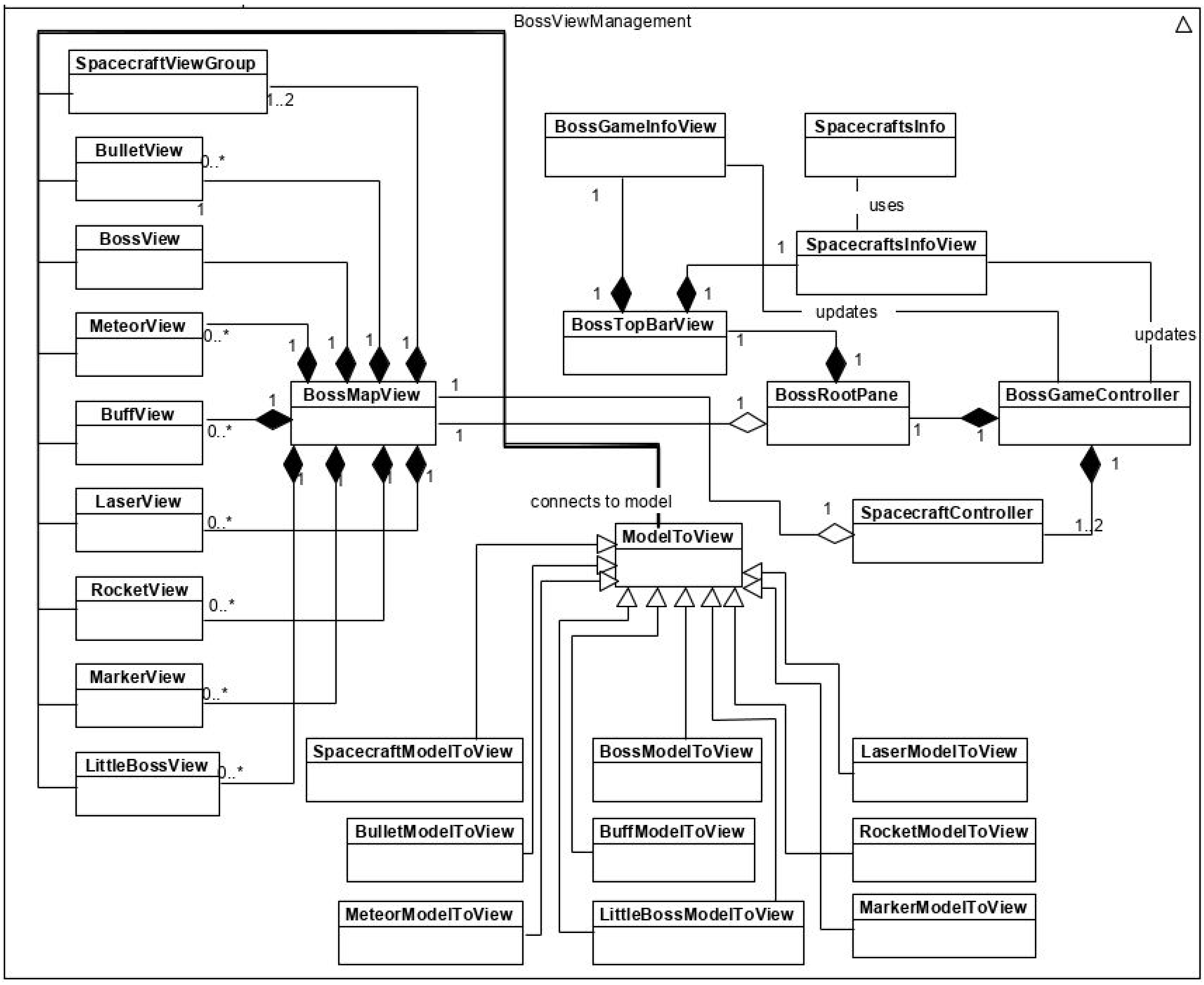
3.3.2. Boss View Management

In Boss View Management, there are view classes which holds views of Boss Scene entities, Boss Map View class to refresh the position of every entity, and controller classes to make connection between view and model classes.

CLASSES IN BOSS VIEW MANAGEMENT

BossMapView Class

This class has functions to refresh the view of the map according to the game entities' locations and actions. The class has aggregation relationships with view class of game entities. It also extends javafx.scene.layout.AnchorPane to put and manage images in the layout.



BossView Class

The class extends javafx.scene.image.ImageView to represent an image. It represents view of the boss.

LaserView Class

The class extends javafx.scene.image.ImageView to represent an image. It represents view of the laser.

RocketView Class

The class extends javafx.scene.image.ImageView to represent an image. It represents view of the rocket.

MarkerView Class

The class extends javafx.scene.image.ImageView to represent an image. It represents view of the marker.

LittleBossView Class

The class extends javafx.scene.image.ImageView to represent an image. It represents view of the little boss.

...ModelToView Classes Related to Boss Subsystem

The functionality of ModelToView class has already been explained in the Pre Boss Subsystem. The followings are added as classes related to only Boss Subsystem.

- **BossModelToView Class**
- **LaserModelToView Class**
- **MarkerModelToView Class**
- **RocketModelToView Class**
- **LittleBossModelToView Class**

BossGameInfoView Class

This class represents the screen view of information about game's current situation such as the player's score.

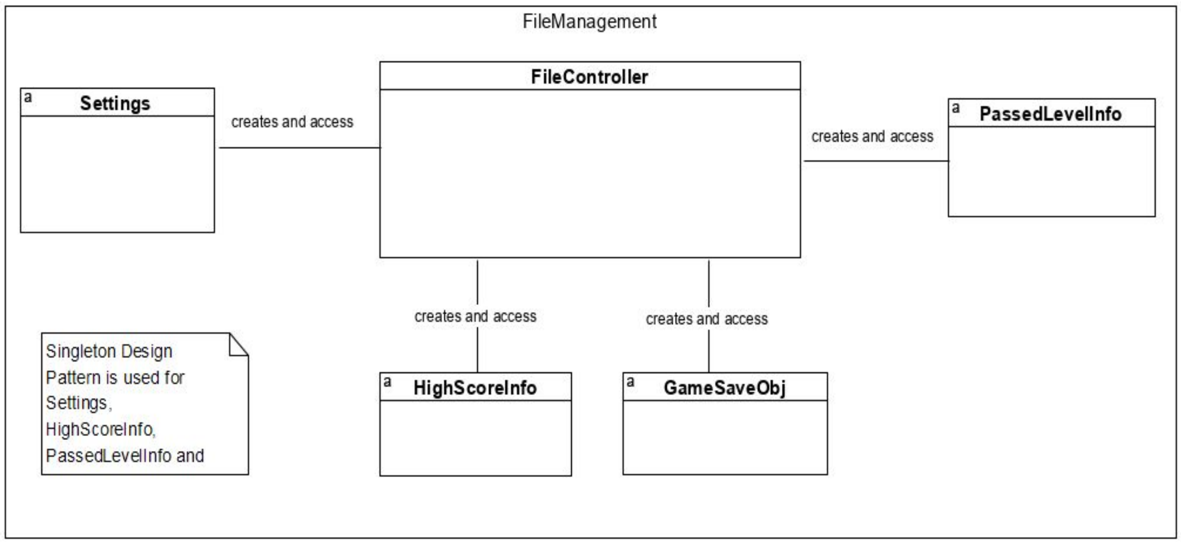
BossTopBarView Class

This class holds SpacecraftInfoView and BossGameInfoView together to handle their layouts.

BossRootPane Class

This class holds BossTopBarView and BossMapView together to handle their layouts.

3.4. File Management Subsystem

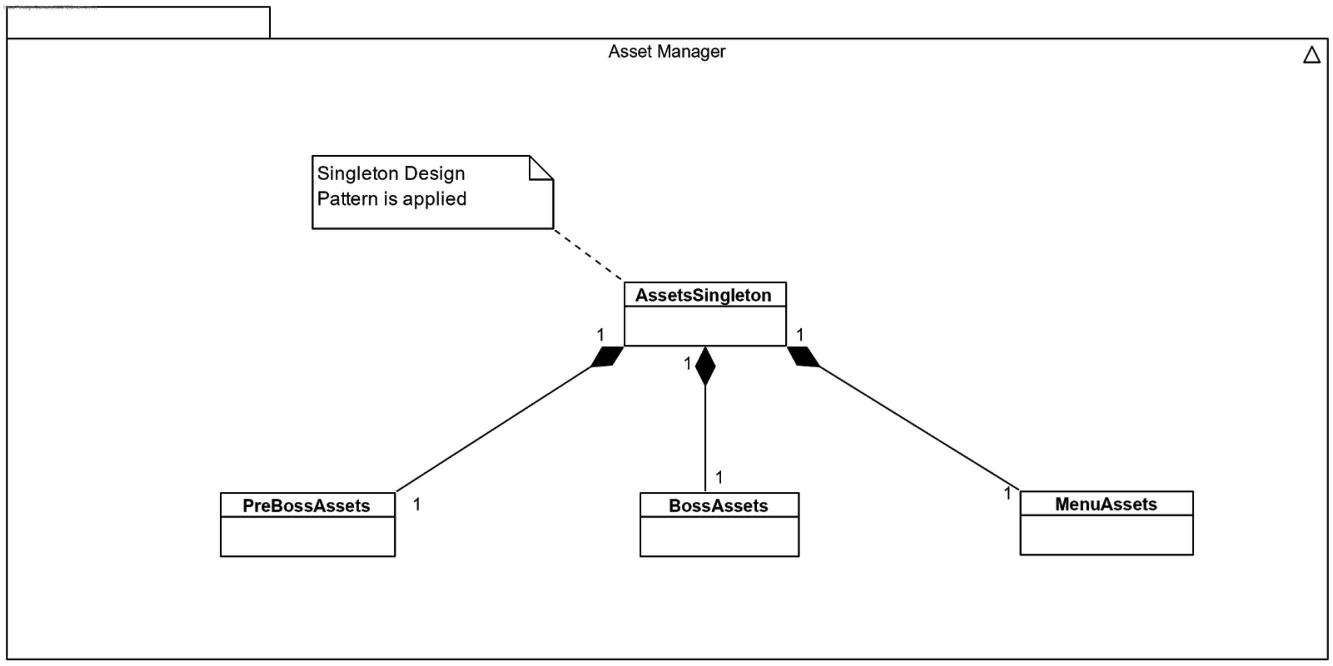


FileController Class:

This class records settings, passed levels, game objects and high scores into the file when its methods are called.

*** Functionalities of other classes are already explained in Menu Subsystem.

3.5. Asset Management Subsystem



MenuAssets

This class holds the assets related to the menu. When the software initializes, this class is created by storing assets from file to its attributes to provide assets to menu classes.

Assets class has 1 instance of this class; therefore, there are 1 to 1 composition relationships between this class and Assets class.

PreBossAssets

This class holds the assets related to the pre boss game stage. When the software initializes, this class is created by storing assets from file to its attributes to provide assets to pre boss game classes.

Assets class has 1 instance of this class; therefore, there are 1 to 1 composition relationships between this class and Assets class.

BossAssets

This class holds the assets related to the boss stage. When the software initialize, this class is created by storing assets from file to its attributes to provide assets to boss classes.

Assets class has 1 instance of this class; therefore, there are 1 to 1 composition relationships between this class and Assets class.

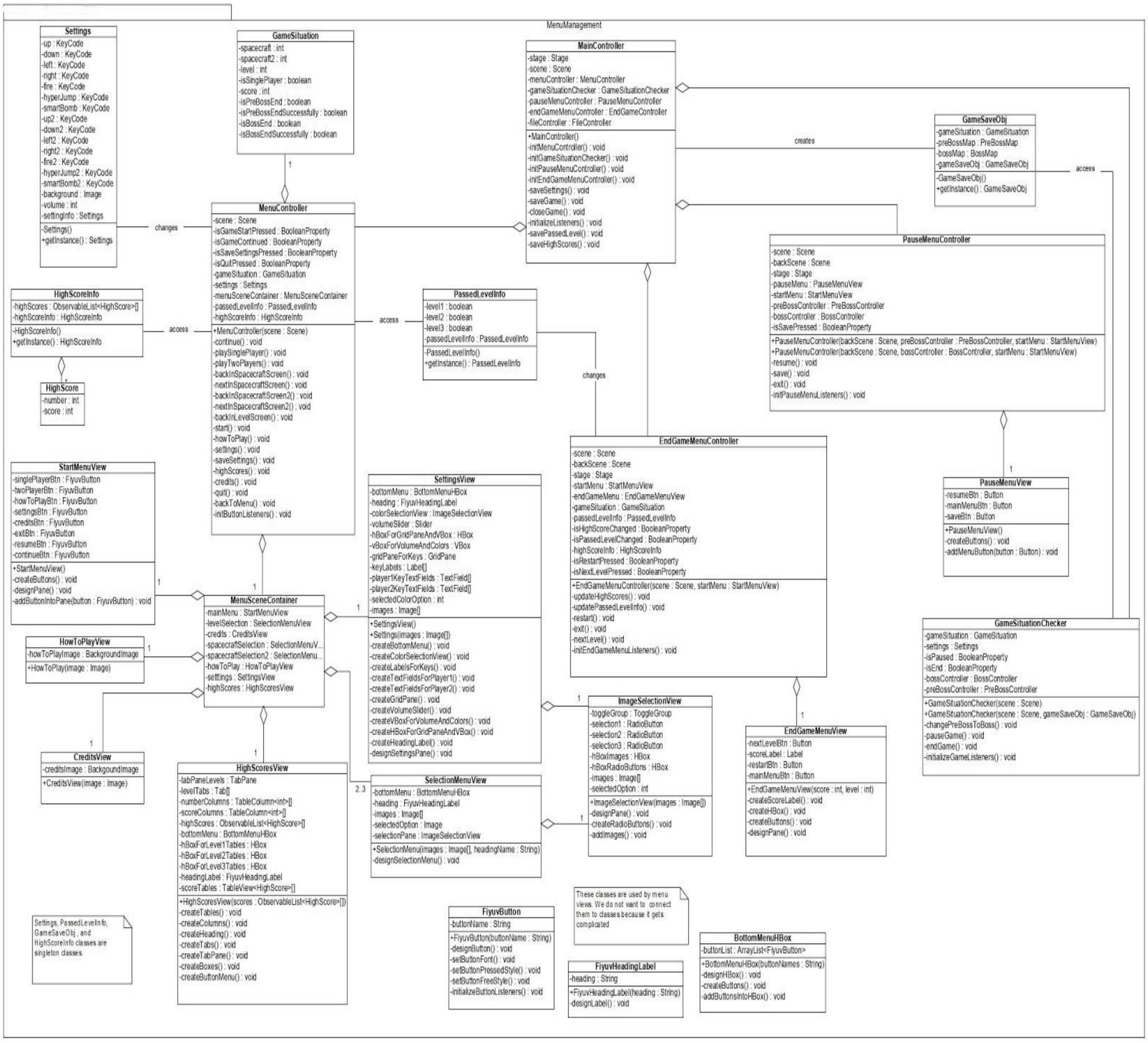
Assets Class

This class is a singleton class which will be created at the beginning of the software to bring all the assets of the game from file in a divided manner.

4. Low Level Design

4.1. Final Object Design

4.1.1. Menu Class Interfaces


MenuManagement
Settings
-up : KeyCode
-down : KeyCode
-left : KeyCode
-right : KeyCode
-fire : KeyCode
-hyperJump : KeyCode
-smartBomb : KeyCode
-up2 : KeyCode
-down2 : KeyCode
-left2 : KeyCode
-right2 : KeyCode
-fire2 : KeyCode
-hyperJump2 : KeyCode
-smartBomb2 : KeyCode
-background : Image
-volume : int
-settingInfo : Settings
-Settings()
+getInstance() : Settings
changes
GameSituation
-spacecraft : int
-spacecraft2 : int
-level : int
-isSinglePlayer : boolean
-score : int
-isPreBossEnd : boolean
-isPreBossEndSuccessfully : boolean
-isBossEnd : boolean
-isBossEndSuccessfully : boolean
1
MainController
-stage : Stage
-scene : Scene
-menuController : MenuController
-gameSituationChecker : GameSituationChecker
-pauseMenuController : PauseMenuController
-endGameMenuController : EndGameController
-fileController : FileController
+MainController()
-initMenuController() : void
-initGameSituationChecker() : void
-initPauseMenuController() : void
-initEndGameMenuController() : void
-saveSettings() : void
-saveGame() : void
-closeGame() : void
-initializeListeners() : void
-savePassedLevel() : void
-saveHighScores() : void
creates
GameSaveObj
-gameSituation : GameSituation
-preBossMap : PreBossMap
-bossMap : BossMap
-gameSaveObj : GameSaveObj
-GameSaveObj()
+getInstance() : GameSaveObj
access
MenuController
-scene : Scene
-isGameStartPressed : BooleanProperty
-isGameContinued : BooleanProperty
-isSaveSettingsPressed : BooleanProperty
-isQuitPressed : BooleanProperty
-gameSituation : GameSituation
-settings : Settings
-menuSceneContainer : MenuSceneContainer
-passedLevelInfo : PassedLevelInfo
-highScoreInfo : HighScoreInfo
+MenuController(scene : Scene)
-continue() : void
-playSinglePlayer() : void
-playTwoPlayers() : void
-backInSpacecraftScreen() : void
-nextInSpacecraftScreen() : void
-backInSpacecraftScreen2() : void
-nextInSpacecraftScreen2() : void
-backInLevelScreen() : void
-start() : void
-howToPlay() : void
-settings() : void
-saveSettings() : void
-highScores() : void
-credits() : void
-quit() : void
-backToMenu() : void
-initButtonListeners() : void
PauseMenuController
-scene : Scene
-backScene : Scene
-stage : Stage
-pauseMenu : PauseMenuView
-startMenu : StartMenuView
-preBossController : PreBossController
-bossController : BossController
-isSavePressed : BooleanProperty
+PauseMenuController(backScene : Scene, preBossController : PreBossController, startMenu : StartMenuView)
+PauseMenuController(backScene : Scene, bossController : BossController, startMenu : StartMenuView)
-resume() : void
-save() : void
-exit() : void
-initPauseMenuListeners() : void
HighScoreInfo
-highScores : ObservableList<HighScore>[]
-highScoreInfo : HighScoreInfo
-HighScoreInfo()
+getInstance() : HighScoreInfo
access
*
HighScore
-number : int
-score : int
access
PassedLevelInfo
-level1 : boolean
-level2 : boolean
-level3 : boolean
-passedLevelInfo : PassedLevelInfo
-PassedLevelInfo()
+getInstance() : PassedLevelInfo
changes
EndGameMenuController
-scene : Scene
-backScene : Scene
-stage : Stage
-startMenu : StartMenuView
-endGameMenu : EndGameMenuView
-gameSituation : GameSituation
-passedLevelInfo : PassedLevelInfo
-isHighScoreChanged : BooleanProperty
-isPassedLevelChanged : BooleanProperty
-highScoreInfo : HighScoreInfo
-isRestartPressed : BooleanProperty
-isNextLevelPressed : BooleanProperty
+EndGameMenuController(scene : Scene, startMenu : StartMenuView)
-updateHighScores() : void
-updatePassedLevelInfo() : void
-restart() : void
-exit() : void
-nextLevel() : void
-initEndGameMenuListeners() : void
1
PauseMenuView
-resumeBtn : Button
-mainMenuBtn : Button
-saveBtn : Button
+PauseMenuView()
-createButtons() : void
-addMenuButton(button : Button) : void
StartMenuView
-singlePlayerBtn : FiyuvButton
-twoPlayerBtn : FiyuvButton
-howToPlayBtn : FiyuvButton
-settingsBtn : FiyuvButton
-creditsBtn : FiyuvButton
-exitBtn : FiyuvButton
-resumeBtn : FiyuvButton
-continueBtn : FiyuvButton
+StartMenuView()
-createButtons() : void
-designPane() : void
-addButtonIntoPane(button : FiyuvButton) : void
1
1
SettingsView
-bottomMenu : BottomMenuHBox
-heading : FiyuvHeadingLabel
-colorSelectionView : ImageSelectionView
-volumeSlider : Slider
-hBoxForGridPaneAndVBox : HBox
-vBoxForVolumeAndColors : VBox
-gridPaneForKeys : GridPane
-keyLabels : Label[]
-player1KeyTextFields : TextField[]
-player2KeyTextFields : TextField[]
-selectedColorOption : int
-images : Image[]
+SettingsView()
+Settings(images : Image[])
-createBottomMenu() : void
-createColorSelectionView() : void
-createLabelsForKeys() : void
-createTextFieldsForPlayer1() : void
-createTextFieldsForPlayer2() : void
-createGridPane() : void
-createVolumeSlider() : void
-createVBoxForVolumeAndColors() : void
-createHBoxForGridPaneAndVBox() : void
-createHeadingLabel() : void
-designSettingsPane() : void
1
MenuSceneContainer
-mainMenu : StartMenuView
-levelSelection : SelectionMenuView
-credits : CreditsView
-spacecraftSelection : SelectionMenuV...
-spacecraftSelection2 : SelectionMenu...
-howToPlay : HowToPlayView
-settings : SettingsView
-highScores : HighScoresView
GameSituationChecker
-gameSituation : GameSituation
-settings : Settings
-isPaused : BooleanProperty
-isEnd : BooleanProperty
-bossController : BossController
-preBossController : PreBossController
+GameSituationChecker(scene : Scene)
+GameSituationChecker(scene : Scene, gameSaveObj : GameSaveObj)
-changePreBossToBoss() : void
-pauseGame() : void
-endGame() : void
-initializeGameListeners() : void
HowToPlayView
-howToPlayImage : BackgroundImage
+HowToPlay(image : Image)
1
1
ImageSelectionView
-toggleGroup : ToggleGroup
-selection1 : RadioButton
-selection2 : RadioButton
-selection3 : RadioButton
-hBoxImages : HBox
-hBoxRadioButtons : HBox
-images : Image[]
-selectedOption : int
+ImageSelectionView(images : Image[])
-designPane() : void
-createRadioButtons() : void
-addImages() : void
1
EndGameMenuView
-nextLevelBtn : Button
-scoreLabel : Label
-restartBtn : Button
-mainMenuBtn : Button
+EndGameMenuView(score : int, level : int)
-createScoreLabel() : void
-createHBox() : void
-createButtons() : void
-designPane() : void
1
CreditsView
-creditsImage : BackgoundImage
+CreditsView(image : Image)
1
HighScoresView
-tabPaneLevels : TabPane
-levelTabs : Tab[]
-numberColumns : TableColumn<int>[]
-scoreColumns : TableColumn<int>[]
-highScores : ObservableList<HighScore>[]
-bottomMenu : BottomMenuHBox
-hBoxForLevel1Tables : HBox
-hBoxForLevel2Tables : HBox
-hBoxForLevel3Tables : HBox
-headingLabel : FiyuvHeadingLabel
-scoreTables : TableView<HighScore>[]
+HighScoresView(scores : ObservableList<HighScore>[])
-createTables() : void
-createColumns() : void
-createHeading() : void
-createTabs() : void
-createTabPane() : void
-createBoxes() : void
-createButtonMenu() : void
2..3
SelectionMenuView
-bottomMenu : BottomMenuHBox
-heading : FiyuvHeadingLabel
-images : Image[]
-selectedOption : Image
-selectionPane : ImageSelectionView
+SelectionMenu(images : Image[], headingName : String)
-designSelectionMenu() : void
1
These classes are used by menu views. We do not want to connect them to classes because it gets complicated
Settings, PassedLevelInfo, GameSaveObj , and HighScoreInfo classes are singleton classes.
FiyuvButton
-buttonName : String
+FiyuvButton(buttonName : String)
-designButton() : void
-setButtonFont() : void
-setButtonPressedStyle() : void
-setButtonFreeStyle() : void
-initializeButtonListeners() : void
FiyuvHeadingLabel
-heading : String
+FiyuvHeadingLabel(heading : String)
-designLabel() : void
BottomMenuHBox
-buttonList : ArrayList<FiyuvButton>
+BottomMenuHBox(buttonNames : String)
-designHBox() : void
-createButtons() : void
-addButtonsIntoHBox() : void

4.1.1.1. Interfaces of Controller Classes

<u>MainController Class</u>

Attributes:

- **private Stage stage:** This is the stage of the game.
- **private Scene scene:** This is the scene of the game. It is used for menu screens and for the game itself.
- **private MenuController menuController:** This is an instance of MenuController class. It is used to provide transition to and from Menu screens. The game is started when isGameStartPressed or isGameContinued properties of menuController changes. Also, settings are recorded in MainController when isSaveSettingPressed property of menuController changes. Moreover, when isQuitPressed property of menuController changes, stage is closed inside MainController.
- **private GameSituationChecker gameSituationChecker:** This instance of GameSituationChecker class is used to provide transition to the game from other screens. It also enables MainController to listen whether the game is paused or whether the game ends or not.
- **private PauseMenuController pauseMenuController:** This instance of PauseMenuController is needed to start Pause Menu when isPaused property of gameSituationChecker changes.

- **private EndGameController endGameMenuController:** This instance of EndGameController is needed to start End Game Menu when isEnd property of gameSituationChecker changes.
- **private FileController fileController:** This instance of FileController is needed to save the game, passed level information, high scores information and settings information into the file.

Constructors:

- **public MainController():** “stage” and “scene” instances are initialized in this constructor. Scene of “stage” is set to “scene” and stage is shown. InitMenuController() and initializeListeners() is called in this constructor.

Methods:

- **private void initMenuController() :** This method initializes menuController and adds change listener to isGameStartPressed, isGameContinued, isSaveSettingsPressed, isQuitPressed properties of menuController. This method is called in the constructor.
- **private void initGameSituationChecker() :** This method initializes gameSituationChecker when isGameStartPressed or isGameContinued properties of menuController has changed and adds change listeners to isPaused and isEnd properties of the gameSituationChecker.

- **private void initPauseMenuController() :** This method initializes pauseMenuController when isPaused property of gameSituationChecker has changed and adds listener to isSavePressed property of pauseMenuController.
- **private void initEndGameMenuController() :** This method initializes endGameMenuController when isEnd property of gameSituationChecker has changed and adds change listener to isHighScoreChanged, isPassedLevelChanged, isRestartPressed, and isNextLevelPressed properties of endGameMenuController.
- **private void saveSettings() :** This method saves settings when isSaveSettingsPressed property of menuController has changed.
- **private void saveGame() :** This method saves game objects.
- **private void closeGame() :** This method closes the stage when isQuitPressed property of menuController has changed.
- **private void savePassedLevel() :** This method saves passed levels information when isPassedLevelChanged property of endGameMenuController changes.
- **private void saveHighScores() :** This method saves high scores when isHighScoreChanged property of endGameMenuController changes.

MenuController Class

Attributes:

- **private Scene scene:** This attribute holds the scene of the game. There will be only one scene throughout the game and classes will be able to change the root of the scene.
- **private BooleanProperty isGameStartPressed:** This attribute is needed to inform MainController when “Start” button is pressed in the Level Selection Screen.
- **private BooleanProperty isGameContinued:** This attribute is needed to inform MainController when the player wants to play a saved version of the game.
- **private BooleanProperty isSaveSettingsPressed:** This attribute is needed to inform MainController when “Save” button is pressed in the Settings Screen.
- **private BooleanPropery isQuitPressed:** This attribute is needed to inform MainController when “Quit” button is pressed in the Start Menu.
- **private GameSituation gameSituation:** This attribute holds singleton object of GameSituation class that contains all information about the game.
- **private Settings settings:** This attribute holds singleton object of **Settings class that contains** settings information of the game such as selected key codes, volume and background color.

- **private PassedLevelInfo passedLevelInfo:** This attribute holds singleton object of PassedLevelInfo class that contains information of passed levels because the player should not select level 3 in the menu to play before passing level 2.
- **private HighScoreInfo highScoreInfo:** This attribute holds singleton object of HighScoreInfo class that contains information about high scores for all levels.
- **private MenuSceneContainer menuSceneContainer:** This attribute holds an object of MenuSceneContainer which contains objects of all menu view classes needed to set the root of scene in order to change screen.

Constructors:

- **public MenuController(Scene scene):** MenuController takes scene of the game to be able to set the root of the game scene to provide transition between screens. In the constructor, the root of the scene set to StartMenuView instance of menuSceneContainer and initButtonListeners() method is called.

Methods:

- **private void continue():** This method sets isGameContinued to true when “Resume” button is pressed in the Start Menu.
- **private void playSinglePlayer():** This method provides transition from the Start Menu to Spacecraft1Selection screen by changing scene root.

Also, it sets the number of players attribute of gameSituation instance to true.

- **private void playTwoPlayers():** This method provides transition from the start menu to Spacecraft1Selection screen by changing scene root. Also, it sets the number of players attribute of gameSituation instance to false.
- **private void nextInSpacecraftScreen():** This method provides transition from Spacecraft1Selection screen to Spacecraft2Selection screen if the game is played by two players. Otherwise, the method provides transition from spacecraft 1 selection screen to level selection screen by changing scene root. Also, it sets the image of spacecraft in GameSituation class to selected image.
- **private void backInSpacecraftScreen():** This method provides transition from Spacecraft1Selection screen to start menu screen by changing scene root.
- **private void nextInSpacecraftScreen2():** This method provides transition from Spacecraft2Selection screen to level selection screen by changing scene root. It also sets the image of spacecraft 2 in GameSituation class to selected image.
- **private void backInSpacecraftScreen2():** This method provides transition from Spacecraft2Selection screen to spacecraft 1 screen by changing root of the scene.

- **private void backInLevelScreen():** This method provides transition from LevelSelection screen to Spacecraft2Selection screen if the number of players is 2. It provides transition from LevelSelection screen to Spacecraft1Selection screen if the number of players is 1.
- **private void start():** This method sets the value of isGameStartPressed property to true.
- **private void howToPlay():** This method provides transition from start menu to HowToPlay screen.
- **private void settings():** This method provides transition from Start Menu to Settings screen.
- **private void saveSettings():** This method is associated with “Save” button in settings screen. It sets the value of isSaveSettingsPressed property to true.
- **private void highScores():** This method provides transition from start menu to HighScores screen.
- **private void credits():** This method provides transition from Start Menu to credits screen.
- **private void quit():** This method sets the value of isQuitPressed property to true.
- **private void backToMenu():** This method provides transition from HowToPlay, Settings, HighScore, and Credits screens to Start Menu.

- **private void initButtonListeners():** This method provides connection between menu buttons and methods above. It gets buttons from menuSceneContainer and calls setOnAction method on them.

GameSituationChecker Class

Attributes:

- **private GameSituation gameSituation:** This attribute holds singleton object of GameSituation class.
- **Settings settings:** This attribute holds singleton object of Settings class.
- **private BooleanProperty isPaused:** This attribute is needed to inform MainController when the game is paused.
- **private BooleanProperty isEnd:** This attribute is needed to inform MainController when the game ends.
- **private BossController bossController:** This attribute is needed to start Boss section of the game.
- **private PreBossController preBossController:** This attribute is needed to start PreBoss section of the game.

Constructors:

- **public GameSituationChecker(Scene scene):** This constructor takes the scene of the game as parameter to be able to transfer it into the constructor of PreBossController or BossController. This constructor is used when a new game is started and it initializes preBossController using gameSituation.

- **public GameSituationChecker(Scene scene, GameSaveObj gameSaveObj):** This constructor takes the scene of the game as parameter to be able to transfer it into the constructor of PreBossController or BossController. This constructor also takes an instance of GameSaveObj in order to initialize recorded game. Inside the constructor, gameSituation inside gameSaveObj is assigned to gameSituation attribute of this class. After this, we look at whether the game is in PreBossScene is finished or not. If it is not finished, we get preBossMap from gameSaveObj and initialize PreBossController with this map. If PreBossScene is finished, we get bossMap from gameSaveObj and initialize BossController with this map.

Methods:

- **private void changePreBossToBoss() :** When PreBoss section of the game finished, this method creates an instance of BossController. Also, it updates isPreBossEnd and isPreBossEndSuccessfully attributes in GameSituation class.
- **private void pauseGame() :** This method sets the value of isPaused property to true.
- **private void endGame() :** This method sets the value of isEnd property to true.

- **private void initializeGameListeners() :** This method initializes change listeners to listen whether pause is pressed, or whether PreBossScene or BossScene finishes from PreBossController and BossController classes. Also, it associates change listeners with related methods above.

EndGameMenuController Class

Attributes:

- **private Stage stage:** This is the new stage created the game has finished.
- **private Scene scene:** This is the new scene created after the game has finished. “stage” will contain this scene.
- **private Scene backScene:** This is the scene of the game. It will stay in the background stage when end game menu is initialized.
- **private StartMenuView startMenu:** This attribute is needed to set the root of the scene of the background stage to the root of Start Menu.
- **private EndGameMenuView endGameMenu:** This attribute is needed to set the root of “scene” to the root of End Game Menu.
- **private GameSituationChecker gameSituationChecker:** This attribute is needed to check whether the game has finished or not.
- **private PassedLevelInfo passedLevelInfo:** This attribute holds singleton object of PassedLevelInfo class. This is needed to update passed level information if needed when the game ends.

- **private HighScoreInfo highScoreInfo:** This attribute holds singleton object of HighScoreInfo class. This is needed to update high scores.
- **private GameSituation gameSituation:** This attribute holds singleton object of GameSituation class. It is needed to check the score of the last game, to reach level information and to check whether the game end successfully or not.
- **private BooleanProperty isHighScoreChanged:** This attribute is needed to inform MainController when high scores change.
- **private BooleanProperty isPassedLevelChanged:** This attribute is needed to inform MainController when passed level information changes.
- **private BooleanProperty isRestartPressed:** This attribute is needed to inform MainController when “Restart” button is pressed in End Game Menu.
- **private BooleanProperty isNextLevelPressed:** This attribute is needed to inform MainController when “Next Level” button is pressed.

Constructors:

- **public EndGameMenuController(Scene backScene, StartMenuView startMenu):** Constructor of this class takes the scene of the game and an instance of StartMenuView as parameter in order to be able to change the root of the game scene when “Exit” is pressed. When “Exit” is pressed, the player returns to Start Menu screen.

Methods:

- **private void updateHighScores() :** This class updates high scores in HighScore class if the last game has finished successfully and its score is higher than one the 10 high scores. When high scores change, it sets the value of isHighScoreChanged to true.
- **private void updatePassedLevelInfo():** This class updates passed level information in PassedLevelInfo class if the game has finished succesfully. When the passed level information changes, it sets the value of isPassedLevelChanged property to true.
- **private void restart():** This method sets the value of isRestartPressed property to true.
- **private void exit():** This method changes the root of the backScene to the root of Start Menu and closes the stage of End Game Menu.
- **private void nextLevel():** This method sets the value of isNextLevelPressed property to true.
- **private void initEndGameMenuListeners() :** This method associates End Game Menu buttons with their related methods. It gets buttons from EndGameMenuView and calls setOnAction method on them.

PauseMenuController Class

Attributes:

- **private Stage stage:** This is the Pause Menu stage created when the game is paused.
- **private Scene scene:** This is the Pause Menu scene. It is put into Pause Menu stage when the game is paused.
- **private Scene backScene:** This is the game scene used for menu scenes and for the game itself.
- **private PauseMenuView pauseMenu:** This attribute contains the root of Pause Menu screen.
- **private StartMenuView startMenu:** This attribute contains the root of Start Menu screen. This attribute is needed to be able to set the root of the backScene when “Exit” is pressed. When “Exit” button is pressed, the player returns to Start Menu.
- **private PreBossController preBossController:** This attribute is needed to start animation timer of PreBossController when “Resume” button is pressed.
- **private BossController bossController:** This attribute is needed to start animation timer of BossController when “Resume” button is pressed.
- **private BooleanProperty isSavePressed:** This attribute is needed to inform MainController when “Save” button is pressed.

Constructors:

- **public PauseMenuController(Scene backScene, PreBossController preBossController, StartMenuView startMenu):** This constructor takes scene of the game, an instance of StartMenuView and an instance of preBossController as parameter in order to be able to set scene of the game when “Exit” button is pressed and to be able to start animation timer of preBossController when “Resume” button is pressed. This constructor is called when pause key is pressed in the PreBoss section of the game.
- **public PauseMenuController(Scene backScene, BossController bossController, StartMenuView startMenu):** This constructor takes scene of the game, an instance of StartMenuView and an instance of preBossController as parameter in order to be able to set scene of the game when “Exit” button is pressed and to be able to start animation timer of bossController when “Resume” button is pressed. This constructor is called when pause key is pressed in the Boss section of the game.

Methods:

- **private void resume():** This method starts animation timer of preBossController.
- **private void save():** This method sets the value of isSavePressed property to true.

- **private void exit():** This method sets the root of the game scene to startMenu.
- **private void initPauseMenuListeners():** This method associates buttons of Pause Menu with related methods. It gets buttons from PauseMenuView class and calls setOnAction method on them.

4.1.1.2. Interfaces of View Classes

MenuSceneContainer Class

Attributes:

- **private StartMenuView mainMenu:** This instance contains the root of Start Menu screen.
- **private levelSelection SelectionMenuView:** This instance contains the root of LevelSelection screen.
- **private CreditsView credits:** This instance contains the root of Credits screen.
- **private SelectionMenuView spacecraftSelection:** This instance contains the root of Spacecraft1Selection screen.
- **private SelectionMenuView spacecraftSelection2:** This instance contains the root of Spacecraft2Selection screen.
- **private HowToPlayView howToPlay:** This instance contains the root of HowToPlay screen.
- **private SettingsView settings:** This instance contains the root of Settings screen.

- **private HighScoresView highScores:** This instance contains the root of HighScores screen.

StartMenuView Class inherits AnchorPane

Attributes:

- **private FiyuvButton singlePlayerBtn:** This is single player game button.
- **private FiyuvButton twoPlayerBtn:** This is two players game button.
- **private FiyuvButton howToPlayBtn:** This is how to play button.
- **private FiyuvButton settingsBtn:** This is settings button.
- **private FiyuvButton credisBtn:** This is credits button.
- **private FiyuvButton exitBtn:** This is exit button.
- **private FiyuvButton resumeBtn:** This is resume button.
- **private FiyuvButton continueBtn:** This is continue button.

Constructors:

- **public StartMenuView():** In the constructor designPane() method is called.

Methods:

- **private void designPane():** This method calls createButtons() method and design size and appearance of this pane.

- **private void createButtons():** This method initializes buttons and adds them into this pane by calling addButtonIntoPane(FiyuvButton button) method.
- **private void addButtonIntoPane(FiyuvButton button):** This method adds given button into this pane.

<u>SelectionMenuView Class inherits VBox</u>

Attributes:

- **private BottomMenuHBox bottomMenu:** This attribute holds bottom menu of Selection Menu which contains “Back” and “Next” buttons.
- **private FiyuvHeadingLabel heading:** This attribute holds heading label of Selection Menu.
- **private images Image[]:** This holds images for selection.
- **private Image selectedOption:** This attribute holds selected image.
- **private ImageSelectionView selectionPane:** This attribute holds the layout for images and radio buttons.

Constructors:

- **public SelectionMenu(Image[] images, String headingName):** The constructor gets images for selection and heading of selection menu and calls designSelectionMenu() method.

Methods:

- **private void designSelectionMenu():** This method initializes heading, selectionPane and bottomMenu, arranges their sizes and adds them into this pane.

ImageSelectionView Class inherits VBox

Attributes:

- **private ToggleGroup toggleGroup:** This attribute is to select only one radio button at one time.
- **private RadioButton selection1:** This is radio button for selection 1.
- **private RadioButton selection2:** This is radio button for selection 2.
- **private RadioButton selection3:** This is radio button for selection 3.
- **private HBox hBoxImages:** This pane will contain images only.
- **private HBox hBoxRadioButtons:** This pane will contain radio buttons only.
- **private Image[] images:** This attribute holds three images.

Constructors:

- **public ImageSelectionView(Image[] images):** This constructor takes selection images as parameter and calls designPane() method.

Methods:

- **desigPane():** This method initializes hBoxImages and hBoxRadioButtons and calls create createRadioButtons() and addImages() methods. It also adds hBoxImages and hBoxRadioButtons into this pane.

- **private void createRadioButtons():** This method initializes radio buttons and adds them into hBoxRadioButtons.
- **private void addImages():** This method adds images into hBoxImages.

<u>HighScoresView Class inherits BorderPane</u>

Attributes:

- **private TabPane tabPaneLevels:** This TabPane is to be able to create level tabs. Because each level will have its own high score list.
- **private Tab[] levelTabs:** This tabs are to be able to open high scores of each level in HighScore the screen.
- **private TableView<HighScore> scoreTables[]:** This array holds score tables for one player and two player game of each level.
- **private TableColumn<int>[] numberColumns:** This column contains ranking of high scores.
- **private TableColumn<int>[] scoreColumns:** This column contains scores.
- **private ObservableList<HighScore>[] highScores:** This lists are to be able to create score tables for one player and two player game of each level.

- **private BottomMenuHBox bottomMenu:** This will hold bottom menu of HighScores screen which contains only “Back” button.
- **private HBox hBoxForLevel1Tables:** This HBox instance will contain score tables of level 1 for one player and two player game.
- **private HBox hBoxForLevel2Tables:** This HBox instance will contain score tables of level 2 for one player and two player game.
- **private HBox hBoxForLevel3Tables:** This HBox instance will contain score tables of level 3 for one player and two player game.
- **private FiyuvHeadingLabel headingLabel:** This headingLabel is for heading of the screen, namely it will contain “HowToPlay” label.

Constructors:

- **public HighScoresView(ObservableList<HighScore>[] scores):** The constructor takes high scores as parameter, assigns it to highScores attribute and calls designHighScorePane() method.

Methods:

- **private void designHighScorePane():** This method calls createTabPane(), createHeading(), and createBottomMenu() methods and adds bottomMenu, heading and tabPaneLevels into this pane.
- **private void createTabPane():** This method calls createTabs() and createHBoxes() method and then sets the content of each tab to one of hboxes.
- **private void createTabs():** This method creates tabs for each level.

- **private void createHBoxes():** This method creates HBox layouts for each level and calls createTables() method, then tables are added into Hboxes. Each HBox pane contains one player and two player high score tables.
- **private void createTables():** This method creates tables, calls createColumns() method and adds columns into tables.
- **private void createColumns():** This method creates number and score column for each score table.
- **private void createHeading():** This method creates heading of this pane.
- **private void createBottomMenu():** This method creates bottom menu pane. This bottom menu contains only “Back” button.

SettingsView Class inherits VBox

Attributes:

- **private BottomMenuHBox bottomMenu:** This attribute holds bottom pane of Settings. It contains only one “Back” button.
- **private FiyuvHeadingLabel heading:** This attribute holds heading of the Settings screen.
- **private ImageSelectionView colorSelectionView**: This attribute holds selection pane for color selection. It contains color images and radio buttons for selection.

- **private Slider volumeSlider:** This is a slider for changing volume.
- **private HBox hBoxForGridPaneAndVBox:** This pane is to design Settings pane.
- **private VBox vBoxForVolumeAndColors:** This pane is to design Settings pane.
- **private GridPane gridPaneForKeys:** This pane will contain labels for the name of keys such as “UP”, “DOWN” and text fields for key selection.
- **private Label[] keyLabels:** This array contains labels for key names.
- **private TextField[] player1KeyTextFields:** This array contains text fields for key selection for player 1.
- **private TextField[] player2KeyTextFields:** This array contains text fields for key selection for player 2.
- **private Image[] images**: This array holds images of colors.

Constructors:

- **public SettingsView(Image[] images):** This constructor takes images of colors as parameter and calls designSettingsPane() method.
- **public SettingsView():** This constructor initializes settings view with given information of Settings class.

Methods:

- **private void designSettingsPane():** This method calls createHeadingLabel(), createBottomMenu(), and createHBoxForGridPaneAndVBox() methods and adds bottomMenu, hBoxForGridPaneAndVBox and heading into this pane.

- **private void createBottomMenu():** This method creates bottom pane of Settings.
- **private void createHBoxForGridPaneAndVBox():** This method initializes hBoxForGridPaneAndVBox, calls createGridPane() and createVBoxForVolumeAndColors() methods and adds gridPaneForKeys and vBoxForVolumeAndColors into hBoxForGridPaneAndVBox.
- **private void createVBoxForVolumeAndColors():** This method initializes vBoxForVolumeAndColors, calls createVolumeSlider() and createColorSelectionView(), and adds volumeSlider and colorSelectionView into vBoxForVolumeAndColors.
- **private void createColorSelectionView():** This method initializes colorSelectionView with images, width and height.
- **private void createVolumeSlider():** This method initializes volumeSlider.
- **private void createGridPane():** This method initializes gridPaneForKeys and calls createLabelsForKeys(), createTextFieldsForPlayer1() and createTextFieldsForPlayer2() methods and adds keyLabels, player1KeyTextFields, and player2KeyTextFields into grid pane.

- **private void createLabelsForKeys**(): This method initializes keyLabels, creates key labels and adds them into keyLabels.
- **private void createTextFieldsForPlayer1():** This method initializes player1KeyTextFields, creates text fields and adds them into player1KeyTextFields.
- **private void createTextFieldsForPlayer2():** This method initializes player2KeyTextFields, creates text fields and adds them into player1KeyTextFields.
- **private void createHeadingLabel():** This method initializes heading.

<u>HowToPlayView Class inherits Pane</u>

Attributes:

- **private BackgroundImage howToPlayImage:** This is an image which contains information about how to play the game.

Constructors:

- **public HowToPlay(Image image):** Constructor takes image as parameter for background and sets background image of this pane.

<u>CreditsView Class inherits Pane</u>

Attributes:

- **private BackgroundImage creditsImage:** This is an image which contains information about contributors of the game.

Constructors:

- **public HowToPlay(Image image):** Constructor takes image as parameter for background and sets background image of this pane.

4.1.1.3. Interfaces of Model Classes

<u>GameSaveObj Class</u>

Attributes:

- **private GameSituation gameSituation:** This is a singleton object that holds information of the game.
- **private BossMap bossMap :** This is the map of BossScene of the game. It is needed to be saved in order to be used if the user wants to continue previous game when s/he reopens the game.
- **private PreBossMap preBossMap :** This is the map of PreBossScene of the game. It is needed to be saved in order to be used if the user wants to continue previous game when s/he reopens the game.

<u>GameSituation Class</u>

Attributes:

- **private int spacecraft**: This is selected image of spacecraft for player 1.

- **private int spacecraft2:** This is selected image of spacecraft for player 2.
- **private int level:** This is currently played level of the game.
- **private boolean isSinglePlayer:** This attribute holds information about in which mode the game is played, one or two player mode.
- **private int score:** This attribute holds current score of the game.
- **private boolean isPreBossEnd:** This attribute holds whether PreBossScene finished or not.
- **private boolean isPreBossEndSuccessfully:** This attribute holds information about whether PreBossScene finished successfully or not.
- **private boolean isBossEnd:** This attribute holds information about whether BossScene end or not.
- **private boolean isBossEndSuccessfully:** This attribute holds information about whether BossScene end successfully or not.
- **private static GameSituation gameSituation :** This is single instance of this class.

Constructors:

- **private GameSituation(): This constructor is private, because the class is singleton. The constructor initializes attributes of the class.**

Methods:

- **public static** GameSituation **getInstance():** This method returns GameSituation if it is not null. If it is null, it initializes GameSituation and then returns it.

<u>Settings Class</u>

Attributes:

- **private KeyCode up:** This is selected up key for player 1.
- **private KeyCode down:** This is selected down key for player 1.
- **private KeyCode left:** This is selected left key for player 1.
- **private KeyCode right:** This is selected right key for player 1.
- **private KeyCode fire:** This is selected fire key for player 1.
- **private KeyCode hyperJump:** This is selected hyper jump key for player 1.
- **private KeyCode smartBomb:** This is selected smart bomb key for player 1.
- **private KeyCode up2:** This is selected left key up key for player 2.
- **private KeyCode down2:** This is selected down key for player 2.
- **private KeyCode left2:** This is selected left key for player 2.
- **private KeyCode right2:** This is selected right key for player 2.
- **private KeyCode fire2:** This is selected fire key for player 2.
- **private KeyCode hyperJump2:** This is selected hyper jump key for player 2.

- **private KeyCode smartBomb2:** This is selected smart bomb key for player 2.
- **private int backgroundColor:** This is selected background color option.
- **private int volume:** This is selected volume.
- **private static Settings settings:** This is single instance of this class.

Constructors:

- **private Settings():** This constructor is private because this class is singleton. The constructor initializes attributes of the class.

Methods:

- **public static Settings getInstance():** This method initializes settings if it is null. This method returns settings.

PassedLevelInfo Class

Attributes:

- **private boolean level1:** This attribute holds information about whether level 1 is passed or not.
- **private boolean level2:** This attribute holds information about whether level 2 is passed or not.
- **private boolean level3:** This attribute holds information about whether level 3 is passed or not.

- **private static PassedLevelInfo passedLevelInfo:** This is single instance of the class.

Constructors:

- **private PassedLevelInfo():** The constructor of this class is private, because the class is singleton. In the constructor all boolean attributes are set to false.

Methods:

- **public static PassedLevelInfo getInstance():** This method returns passedLevelInfo if it is not null. Otherwise, it initializes passedLevelInfo and then returns passedLevelInfo.

HighScoreInfo Class

Attributes:

- **private ObservableList<HighScore> highScores[]:** This array holds lists of one and two player games for each level.
- **private static HighScoreInfo highScoreInfo:** This is single instance of the class.

Constructor:

- **private HighScoreInfo():** The constructor is private, because this is a singleton class.

Methods

- **public static HighScoreInfo getInstance():** This method initializes highScoreInfo if it is null. This method returns highScoreInfo.

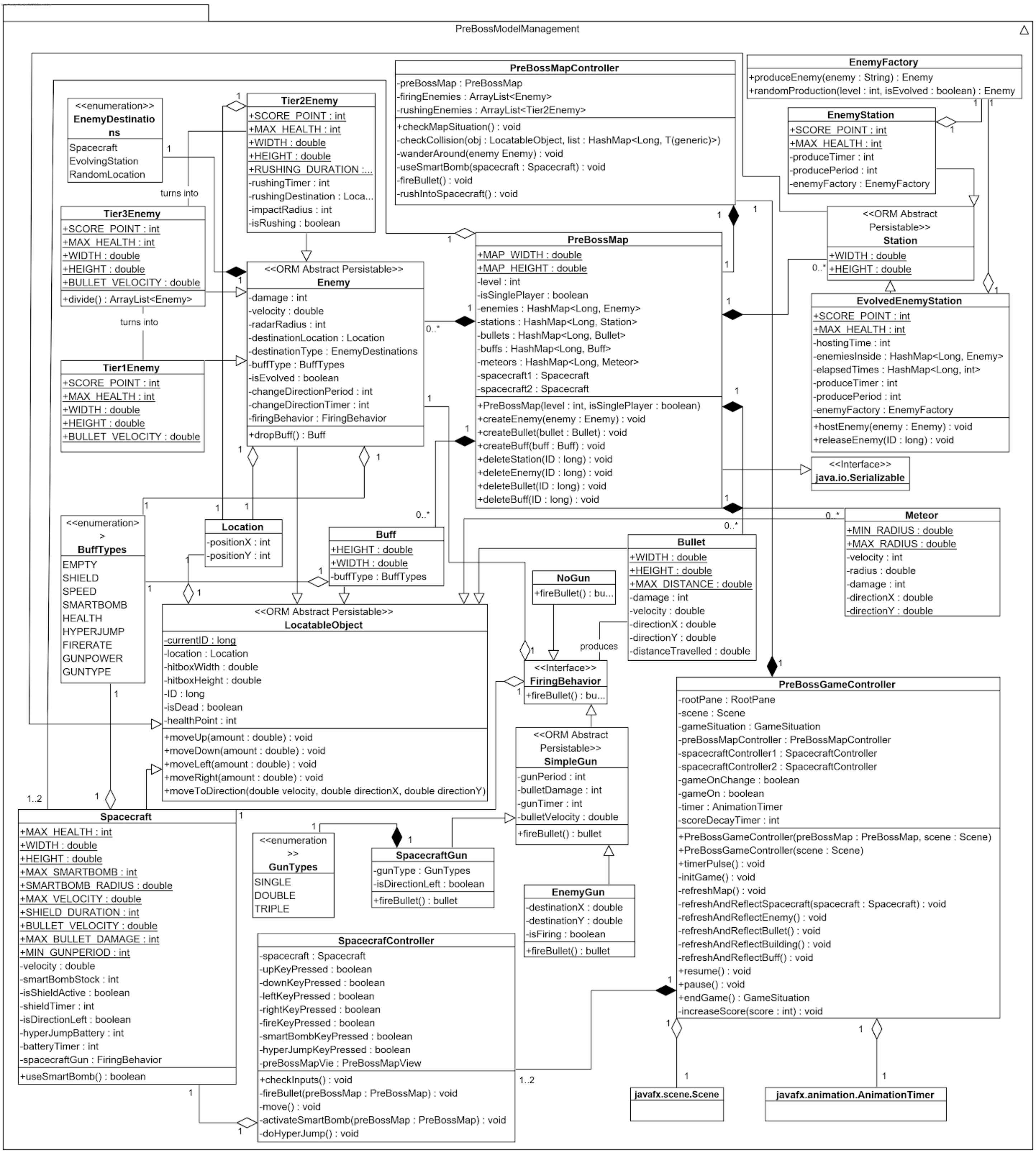
HighScore Class

Attributes:

- **private int number:** This is the information about the ranking.
- **private int score:** This holds the game score.

4.1.2. Pre Boss Scene Class Interfaces

4.1.2.1. Pre Boss Model Management Class Interfaces


PreBossModelManagement
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
<<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
<<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
turns into
Tier3Enemy
+SCORE_POINT : int
+MAX_HEALTH : int
+WIDTH : double
+HEIGHT : double
+BULLET_VELOCITY : double
+divide() : ArrayList<Enemy>
turns into
Tier1Enemy
+SCORE_POINT : int
+MAX_HEALTH : int
+WIDTH : double
+HEIGHT : double
+BULLET_VELOCITY : double
<<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
<<Interface>>
java.io.Serializable
Meteor
+MIN_RADIUS : double
+MAX_RADIUS : double
-velocity : int
-radius : double
-damage : int
-directionX : double
-directionY : double
<<enumeration>>
BuffTypes
EMPTY
SHIELD
SPEED
SMARTBOMB
HEALTH
HYPERJUMP
FIRERATE
GUNPOWER
GUNTYPE
Location
-positionX : int
-positionY : int
Buff
+HEIGHT : double
+WIDTH : double
-buffType : BuffTypes
Bullet
+WIDTH : double
+HEIGHT : double
+MAX_DISTANCE : double
-damage : int
-velocity : double
-directionX : double
-directionY : double
-distanceTravelled : double
NoGun
+fireBullet() : bu...
produces
<<Interface>>
FiringBehavior
+fireBullet() : bu...
<<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)
<<ORM Abstract Persistable>>
SimpleGun
-gunPeriod : int
-bulletDamage : int
-gunTimer : int
-bulletVelocity : double
+fireBullet() : bullet
EnemyGun
-destinationX : double
-destinationY : double
-isFiring : boolean
+fireBullet() : bullet
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
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
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
javafx.scene.Scene
javafx.animation.AnimationTimer
1
0..*
1..2

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 health point 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 EnemyDestinations destinationType:** This attribute holds the type of target point such as spacecraft, enemy stations, random point. These types are held as EnemyDestinations enum.
- **private BuffTypes 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. These buff types are held as BuffTypes enum.
- **private boolean isEvolved:** This attribute holds whether the enemy is evolved or not.
- **private int changeDirectionPeriod:** This attribute holds the period for changing random position direction for wandering around.
- **private int changeDirectionTimer:** This attribute holds the timer for changing random position direction for wandering around.
- **private FiringBehavior firingBehavior:** This attribute holds a firing behavior for enemies. For example, Tier1 and Tier3 enemies will have firing ability, whereas Tier2 enemies won’t have.

Methods:

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

Tier1Enemy Class inherits Enemy

Attributes:

- **public static final double WIDTH:** This attribute holds the hitbox width specific to Tier 1 enemies of both evolved and unevolved.
- **public static final double HEIGHT:** This attribute holds the hitbox height specific to Tier 1 enemies of both evolved and unevolved.
- **public static final double BULLET_VELOCITY:** This attribute holds the hitbox width specific to Tier 1 enemies of both evolved and unevolved.
- **public static final int SCORE_POINT:** This attribute holds the score that will be given after the death of unevolved Tier 1 enemies. For evolved Tier 1 enemies two times of this value is used.
- **public static final int MAX_HEALTH:** This attribute holds the maximum health that will initialized at the beginning of the game for unevolved Tier 1 enemies. For evolved Tier 1 enemies two times of this value is used.

  The attributes above are also used for Tier 2 and Tier 3 enemies as same way. In order to avoid repetition, they are not explained in the other sections.

Tier2Enemy Class inherits Enemy Class

Attributes:

- **public static final double WIDTH**
- **public static final double HEIGHT**
- **public static final int SCORE_POINT**
- **public static final int MAX_HEALTH**
- **public static final int RUSHING_DURATION:** This attribute holds duration time for rushing action of tier 2 enemies.
- **private int rushingTimer:** This attribute holds timer for current rushing action.
- **private Location rushingDestination:** This attribute holds the location of current rushing action.
- **private boolean isRushing:** This attribute holds whether the object is in rushing action or not.
- **private int impact:** This attribute holds radius of Tier 2 enemies' collapsing impact when they crash to spacecraft.

Tier3Enemy Class inherits Enemy, implements FireBullet

Attributes:

- **public static final double WIDTH**
- **public static final double HEIGHT**
- **public static final double BULLET_VELOCITY**
- **public static final int SCORE_POINT**
- **public static final int MAX_HEALTH**

Methods:

- **public ArrayList<Enemy> divide():** This method 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 double WIDTH:** This attribute holds the hitbox width specific to Bullet objects.
- **public static final double HEIGHT:** This attribute holds the hitbox height specific to Bullet objects.
- **public static final double MAX_DISTANCE:** This attribute holds the max distance for one Bullet object to travel.

- **private int damage:** This attribute holds the damage of the Bullet object.
- **private int velocity:** This attribute holds the velocity of the Bullet object.
- **private double directionX:** This attribute holds the X direction of the Bullet object.
- **private double directionY:** This attribute holds the Y direction of the Bullet object.
- **private double distanceTravelled:** This attribute holds the distance on the map that the Bullet object has travelled. When it reaches to max distance the Bullet object dies.

Spacecraft Class inherits LocatableObject

Attributes:

- **public static final double WIDTH:** This attribute holds the hitbox width specific to Spacecraft objects.
- **public static final double HEIGHT:** This attribute holds the hitbox height specific to Spacecraft objects.
- **public static final int MAX_HEALTH:** This attribute holds the maximum health that will initialized at the beginning of the game for Spacecraft objects.

- **public static final int MAX_SMARTBOMB:** This attribute holds the upper boundary of smart bomb slots.
- **public static final double SMARTBOMB_RADIUS:** This attribute holds the impact radius of smart bombs of spacecrafts.
- **public static final double MAX_VELOCITY:** This attribute holds the upper velocity boundary to Spacecraft objects.
- **public static final int SHIELD_DURATION:** This attribute holds the duration time for shield buff for Spacecraft objects.
- **public static final double BULLET_VELOCITY:** This attribute holds the upper velocity boundary to Spacecraft objects’ bullets.
- **public static final double MAX_BULLET_DAMAGE:** This attribute holds the upper damage boundary to Spacecraft objects’ bullets.
- **public static final int MIN_GUNPERIOD:** This attribute holds the lower period boundary to Spacecraft objects’ firing behavior.
- **private double velocity:** This attribute holds the velocity of the Spacecraft.
- **private int smartBombStock:** This attribute holds the current numbers of the smart bombs in the Spacecraft’s stock.
- **private int hyperJumpBattery:** This attribute holds the current percentage of hyper jump battery.
- **private int batteryTimer:** This attribute holds the timer for the hyper jump battery increase. The battery is constantly increase until it reaches to 100%.

- **private boolean isShieldActive:** This attribute holds whether the shield buff is active currently.
- **private int shieldTimer:** This attribute holds the timer for the current shield buff of the Spacecraft.
- **private boolean isDirectionLeft:** This attribute holds the direction of the Spacecraft.
- **private FiringBehavior spacecraftGun:** This attribute holds the gun behavior of Spacecraft object.

Methods:

- **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.

FiringBehavior Interface

Methods:

- **public Bullet fireBullet():** This method needs to be implemented to fire bullets.

SimpleGun Class (abstract) implements FiringBehavior

Attributes:

- **private int bulletDamage:** This attribute holds the damage of the bullets to fire.
- **private int bulletVelocity:** This attribute holds the velocity of the bullets to fire.
- **private int gunPeriod:** This attribute holds the period time for bullets to fire.
- **private int gunTimer:** This attribute holds the current timer to fire next bullet.

Methods:

- **public abstract Bullet fireBullet():** This method needs to be implemented to fire bullets by the subclasses of this class.

EnemyGun Class inherits SimpleGun

Attributes:

- **private double directionX:** This attribute holds the x value directions of bullets to fire.

- **private double directionY:** This attribute holds the y value directions of bullets to fire.
- **private boolean isFiring:** This attribute holds whether the gun is in bullet firing action.

Methods:

- **public abstract Bullet fireBullet():** This method creates a bullet to be put on the map according to the attributes.

<u>SpacecraftGun Class inherits SimpleGun</u>

Attributes:

- **private GunTypes gunType:** This attribute holds the types of spacecraft gun such as single, double, triple as enum.
- **private boolean isDirectionLeft:** This attribute holds the position that the gun is pointed to.

Methods:

- **public abstract Bullet fireBullet():** This method creates a bullet to be put on the map according to the attributes.

NoGun Class implements FiringBehavior

Methods:

- **public abstract Bullet fireBullet():** This method return null for not to fire any bullet.

Meteor Class inherits LocatableObject

Attributes:

- **public static final double MIN_RADIUS:** This attribute holds the lower boundary for radius of a Meteor object.
- **public static final double MAX_RADIUS:** This attribute holds the upper boundary for radius of a Meteor object.
- **private int damage:** This attribute holds the damage of the Meteor object.
- **private int velocity:** This attribute holds the velocity of the Meteor object.
- **private double directionX:** This attribute holds the X direction of the Meteor object.
- **private double directionY:** This attribute holds the Y direction of the Meteor object.

- **private double radius:** This attribute holds the hitbox radius of the Meteor object.

Buff Class inherits LocatableObject

Attributes:

- **public static final double WIDTH:** This attribute holds the hitbox width specific to Buff objects.
- **public static final double HEIGHT:** This attribute holds the hitbox height specific to Buff objects.
- **private BuffTypes buffType:** This attribute holds the type of the Buff as enum.

Station Class (Abstract) inherits LocatableObject

Attributes:

- **public static final double WIDTH:** This attribute holds the hitbox width specific to Station objects.
- **public static final double HEIGHT:** This attribute holds the hitbox height specific to Station objects.

EnemyStation Class inherits Station

Attributes:

- **public static final int SCORE_POINT:** This attribute holds the score that will be given after the destruction of Enemy Stations.
- **public static final int MAX_HEALTH:** This attribute holds the maximum health that will initialized at the beginning of the game for Enemy Stations.
- **private int producePeriod:** This attribute holds the production period of Enemy Station object.
- **private int produceTimer:** This attribute holds the current production timer of Enemy Station object.
- **private EnemyFactory enemyFactory:** This attribute to use factory for producing enemies.

EvolvedEnemyStation Class inherits Station

Attributes:

- **public static final int SCORE_POINT:** This attribute holds the score that will be given after the destruction of Evolved Enemy Stations.

- **public static final int MAX_HEALTH:** This attribute holds the maximum health that will initialized at the beginning of the game for Evolved Enemy Stations.
- **public int hostingTime:** This attribute holds the duration of evaluation process for Evolved Enemy Station object.
- **private int producePeriod:** This attribute holds the production period of Evolved Enemy Station object.
- **private int produceTimer:** This attribute holds the current production timer of Evolved Enemy Station object.
- **private HashMap<Long, Enemy> enemiesInside:** This is the list of enemies currently evolving in this station.
- **private HashMap<Long, int> elapsedTime:** This is the timer list for enemies currently evolving in this station. When time of any enemy reaches to hostingTime, it will be released as evolved.
- **private EnemyFactory enemyFactory:** This attribute to use factory for producing enemies.

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.

EnemyFactory Class

Methods:

- **public Enemy produceEnemy(String enemy):** This method creates a new Enemy according to the given parameter.
- **public Enemy randomProduction(int level, boolean isEvolved):** This method creates a random enemy according to the given parameters.

PreBossMap Class implements (Java interface) Serializable

This class represents the current situation of the map. Where the stations, 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:

- **public static final double MAP_WIDTH:** This attribute holds the width value of the map.
- **public static final double MAP_HEIGHT:** This attribute holds the height value of the map.
- **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 boolean isSinglePlayer:** This attribute holds whether current game is single player or not.
- **private HashMap<Long, Enemy> enemies:** This attribute keeps track of the enemies the map contains.
- **private HashMap<Long, Station> stations:** This attribute keeps track of the stations 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 Spacecraft spacecraft1:** 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.

Methods:

- **public PreBossMap(int level, boolean isSinglePlayer):** This constructor initializes a map with given parameters.
- **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 deleteStation(Long ID):** This method destroys station 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.

PreBossMapController Class

Attributes:

- **private PreBossMap preBossMap:** This is an instance of the Pre Boss Map model.

- **private ArrayList<Enemy> firingEnemies:** This attribute holds the enemies that is in a firing situation to a spacecraft.
- **private ArrayList<Tier2Enemy> rushingEnemies:** This attribute holds the Tier 2 enemies that are in a rushing situation to a spacecraft.

Methods:

- **public void checkMapSituation():** This method is to manipulate map according to its current situation. This method is called at every timer pulse ,except game is in pause state, by PreBossGameController. This method calls other methods of this controller to handle business logic.
- **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 random movement of given enemies if they don’t have any spacecraft in their radar.
- **private void useSmartBomb(Spacecraft spacecraft):** This method applies the effects of the smart bomb used by the given spacecraft on the map.
- **private void fireBullet():** This method applies the bullet firing actions for enemies that are in the firingEnemies list.

- **private void rushIntoSpacecraft():** This method applies the rushing into a spacecraft actions for Tier 2 enemies that are in the rushingEnemies list.

PreBossGameController Class

Attributes:

- **private RootPane rootPane:**
- **private Scene scene:**

  The attributes above are explained in view related section to separate concerns.

- **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 PreBossMapController preBossMapController:** This attribute is to build the connection between entities of the game to the Pre Boss Game Controller via Pre Boss Map Controller to handle business logic.
- **private SpacecraftController spacecraftController1:** This attribute is to handle business logic of the first spacecraft separately from other concerns.

- **private SpacecraftController spacecraftController2:** This attribute is to handle business logic of the second spacecraft, if the game is not single player mode, separately from other concerns.
- **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 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 AnimationTimer animationTimer:** This attribute is used to create time pulses for continuity of the game. The animationTimer will stop when the game in pause stage and will play if the game is on.
- **private int scoreDecayTimer:** This attribute is to provide a timer for score decay logic.

Attributes:

- **PreBossGameController(Scene scene):** This is a constructor to build up the game to given scene object.
- **PreBossGameController(PreBossMap preBossMap, Scene scene):** 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 timerPulse():** This method is called at every time pulse created by AnimationTimer object of JavaFx and every business logic related to the time is invoked either directly or indirectly from here.
- **private void refreshMap():** This method is to refresh map at every timer pulse. Refreshing includes actions such as checking collisions, setting destinations for enemies, location changes in objects.
- **private void refreshAndReflectEnemy():**
- **private void refreshAndReflectBullet():**
- **private void refreshAndReflectSpacecraft(Spacecraft spacecraft):**
- **private void refreshAndReflectMeteor():**
- **private void refreshAndReflectStation():**
- **private void refreshAndReflectBuff():**

  These method above are to reflect the changes in the Pre Boss Map 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 Pre Boss Map.*

- **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.

SpacecraftController Class

Attributes:

- **private PreBossMapView preBossMapView:**

  This attributes is explained in view related section to separate concerns.

- **private Spacecraft spacecraft:** This attribute holds a spacecraft object to be controlled by this class.
- **private boolean upKeyPressed:** This attribute holds whether the specific key is pressed in order to apply going up motion.
- **private boolean downKeyPressed:** This attribute holds whether the specific key is pressed in order to apply going down motion.

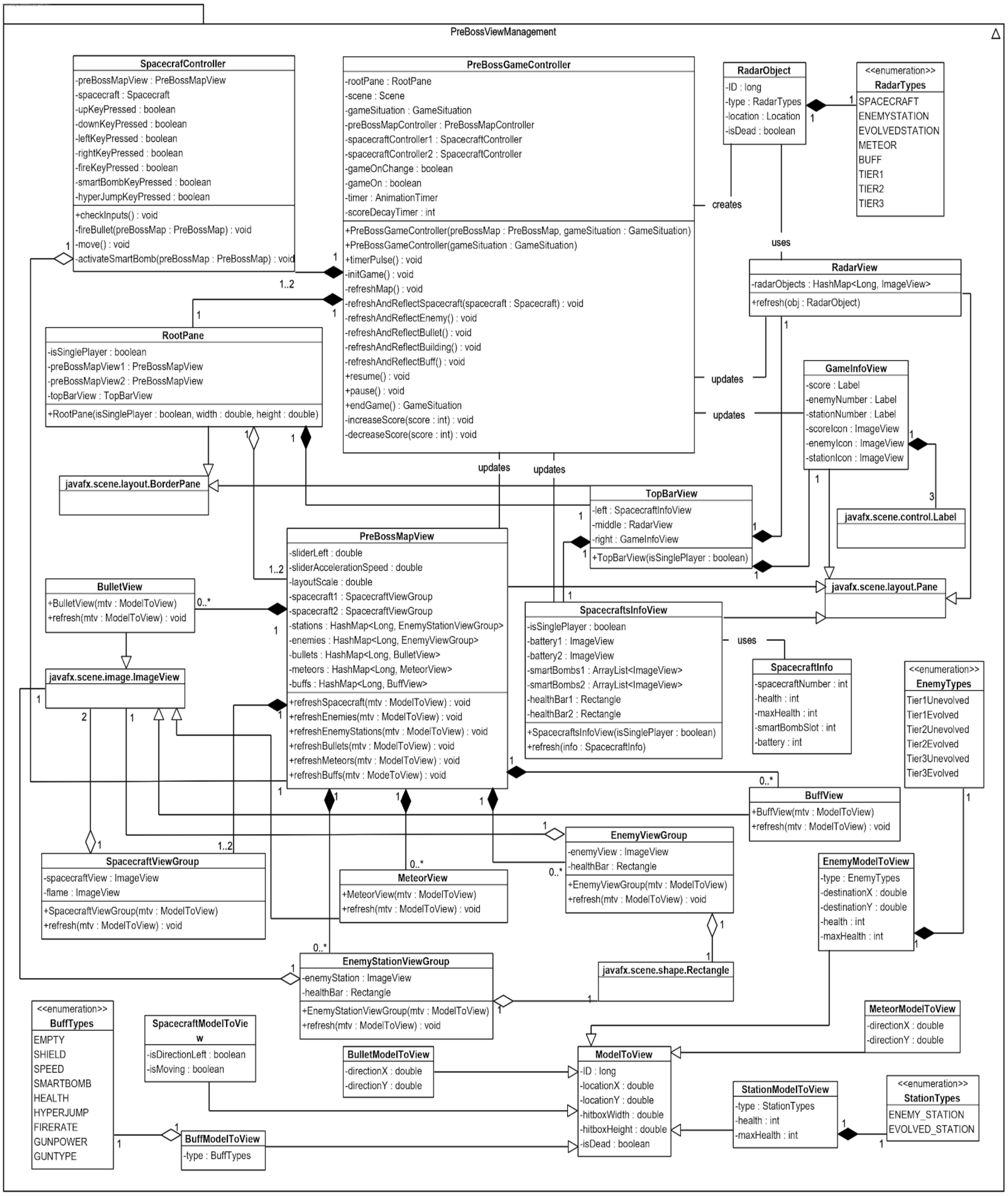
- **private boolean leftKeyPressed:** This attribute holds whether the specific key is pressed in order to apply going left motion.
- **private boolean rightKeyPressed:** This attribute holds whether the specific key is pressed in order to apply going right motion.
- **private boolean fireKeyPressed:** This attribute holds whether the specific key is pressed in order to apply fire bullet method for Spacecraft.
- **private boolean hyperJumpKeyPressed:** This attribute holds whether the specific key is pressed in order to apply hyper jump if battery is charged enough.
- **private boolean smartBombKeyPressed:** This attribute holds whether the specific key is pressed in order to apply smart bomb if there is any in the stock.

Methods:

- **public void checkInputs():** This method is to check the current situation of keys pressed and apply their consequences. This method is called at every timer impulse by PreBossGameController class.
- **private void fireBullet(PreBossMap preBossMap):** This method is to fire a bullet on the map given which will be the same map that our spacecraft is on as well.
- **private void move():** This method is to move spacecraft according to the movement keys pressed.

- **private void activateSmartBomb(PreBossMap preBossMap):** This method is activate a smart bomb on the map given by calling its smart bomb related method by sending this spacecraft as parameter.
- **private void doHyperJump():** This method is to do a hyper jump.

4.1.2.2 Pre Boss View Management Class Interfaces


PreBossViewManagement
SpacecrafController
-preBossMapView : PreBossMapView
-spacecraft : Spacecraft
-upKeyPressed : boolean
-downKeyPressed : boolean
-leftKeyPressed : boolean
-rightKeyPressed : boolean
-fireKeyPressed : boolean
-smartBombKeyPressed : boolean
-hyperJumpKeyPressed : boolean
+checkInputs() : void
-fireBullet(preBossMap : PreBossMap) : void
-move() : void
-activateSmartBomb(preBossMap : PreBossMap) : void
PreBossGameController
-rootPane : RootPane
-scene : Scene
-gameSituation : GameSituation
-preBossMapController : PreBossMapController
-spacecraftController1 : SpacecraftController
-spacecraftController2 : SpacecraftController
-gameOnChange : boolean
-gameOn : boolean
-timer : AnimationTimer
-scoreDecayTimer : int
+PreBossGameController(preBossMap : PreBossMap, gameSituation : GameSituation)
+PreBossGameController(gameSituation : GameSituation)
+timerPulse() : void
-initGame() : void
-refreshMap() : void
-refreshAndReflectSpacecraft(spacecraft : Spacecraft) : void
-refreshAndReflectEnemy() : void
-refreshAndReflectBullet() : void
-refreshAndReflectBuilding() : void
-refreshAndReflectBuff() : void
+resume() : void
+pause() : void
+endGame() : GameSituation
-increaseScore(score : int) : void
-decreaseScore(score : int) : void
RadarObject
-ID : long
-type : RadarTypes
-location : Location
-isDead : boolean
<<enumeration>>
RadarTypes
SPACECRAFT
ENEMYSTATION
EVOLVEDSTATION
METEOR
BUFF
TIER1
TIER2
TIER3
creates
uses
RadarView
-radarObjects : HashMap<Long, ImageView>
+refresh(obj : RadarObject)
RootPane
-isSinglePlayer : boolean
-preBossMapView1 : PreBossMapView
-preBossMapView2 : PreBossMapView
-topBarView : TopBarView
+RootPane(isSinglePlayer : boolean, width : double, height : double)
updates
updates
GameInfoView
-score : Label
-enemyNumber : Label
-stationNumber : Label
-scoreIcon : ImageView
-enemyIcon : ImageView
-stationIcon : ImageView
javafx.scene.layout.BorderPane
updates
updates
TopBarView
-left : SpacecraftInfoView
-middle : RadarView
-right : GameInfoView
+TopBarView(isSinglePlayer : boolean)
javafx.scene.control.Label
PreBossMapView
-sliderLeft : double
-sliderAccelerationSpeed : double
-layoutScale : double
-spacecraft1 : SpacecraftViewGroup
-spacecraft2 : SpacecraftViewGroup
-stations : HashMap<Long, EnemyStationViewGroup>
-enemies : HashMap<Long, EnemyViewGroup>
-bullets : HashMap<Long, BulletView>
-meteors : HashMap<Long, MeteorView>
-buffs : HashMap<Long, BuffView>
+refreshSpacecraft(mtv : ModelToView) : void
+refreshEnemies(mtv : ModelToView) : void
+refreshEnemyStations(mtv : ModelToView) : void
+refreshBullets(mtv : ModelToView) : void
+refreshMeteors(mtv : ModelToView) : void
+refreshBuffs(mtv : ModeToView) : void
BulletView
+BulletView(mtv : ModelToView)
+refresh(mtv : ModelToView) : void
javafx.scene.layout.Pane
SpacecraftsInfoView
-isSinglePlayer : boolean
-battery1 : ImageView
-battery2 : ImageView
-smartBombs1 : ArrayList<ImageView>
-smartBombs2 : ArrayList<ImageView>
-healthBar1 : Rectangle
-healthBar2 : Rectangle
+SpacecraftsInfoView(isSinglePlayer : boolean)
+refresh(info : SpacecraftInfo)
uses
SpacecraftInfo
-spacecraftNumber : int
-health : int
-maxHealth : int
-smartBombSlot : int
-battery : int
<<enumeration>>
EnemyTypes
Tier1Unevolved
Tier1Evolved
Tier2Unevolved
Tier2Evolved
Tier3Unevolved
Tier3Evolved
javafx.scene.image.ImageView
BuffView
+BuffView(mtv : ModelToView)
+refresh(mtv : ModelToView) : void
EnemyViewGroup
-enemyView : ImageView
-healthBar : Rectangle
+EnemyViewGroup(mtv : ModelToView)
+refresh(mtv : ModelToView) : void
SpacecraftViewGroup
-spacecraftView : ImageView
-flame : ImageView
+SpacecraftViewGroup(mtv : ModelToView)
+refresh(mtv : ModelToView) : void
MeteorView
+MeteorView(mtv : ModelToView)
+refresh(mtv : ModelToView) : void
EnemyModelToView
-type : EnemyTypes
-destinationX : double
-destinationY : double
-health : int
-maxHealth : int
EnemyStationViewGroup
-enemyStation : ImageView
-healthBar : Rectangle
+EnemyStationViewGroup(mtv : ModelToView)
+refresh(mtv : ModelToView) : void
javafx.scene.shape.Rectangle
<<enumeration>>
BuffTypes
EMPTY
SHIELD
SPEED
SMARTBOMB
HEALTH
HYPERJUMP
FIRERATE
GUNPOWER
GUNTYPE
SpacecraftModelToView
-isDirectionLeft : boolean
-isMoving : boolean
MeteorModelToView
-directionX : double
-directionY : double
BulletModelToView
-directionX : double
-directionY : double
ModelToView
-ID : long
-locationX : double
-locationY : double
-hitboxWidth : double
-hitboxHeight : double
-isDead : boolean
StationModelToView
-type : StationTypes
-health : int
-maxHealth : int
<<enumeration>>
StationTypes
ENEMY_STATION
EVOLVED_STATION
BuffModelToView
-type : BuffTypes
1
1..2
0..*
3
2
1..2

SpacecraftViewGroup Class

Attributes:

- **private ImageView spacecraftView:** This attribute holds the view of the spacecraft.
- **private ImageView flame:** This attribute holds the view of spacecraft flame when it moves.

Methods:

- **public void SpacecraftViewGroup(ModelToView mtv):** This constructor creates view related to spacecraft according to the its initial model.
- **public void refresh(ModelToView mtv):** This method refreshes view related to spacecraft according to the current information of its model.

EnemyStationViewGroup Class

Attributes:

- **private ImageView enemyStation:** This attribute holds the view of the enemy stations.
- **private Rectangle healthBar:** This attribute holds the view of the health point of enemy stations.

Methods:

- **public void EnemyStationViewGroup(ModelToView mtv):** This constructor creates view related to enemy station according to the its initial model.
- **public void refresh(ModelToView mtv):** This method refreshes view related to enemy station according to the current information of its model.

EnemyViewGroup Class

Attributes:

- **private ImageView enemyView:** This attribute holds the view of the enemies.
- **private Rectangle healthBar:** This attribute holds the view of the health point of enemies.

Methods:

- **public void EnemyViewGroup(ModelToView mtv):** This constructor creates view related to enemy according to the its initial model.
- **public void refresh(ModelToView mtv):** This method refreshes view related to spacecraft according to the current information of its model.

MeteorView Class inherits JavaFx ImageView

Methods:

- **public void MeteorView(ModelToView mtv):** This constructor creates view related to meteor according to the its initial model.
- **public void refresh(ModelToView mtv):** This method refreshes view related to meteor according to the current information of its model.

BuffView Class inherits JavaFx ImageView

Methods:

- **public void BuffView(ModelToView mtv):** This constructor creates view related to buff according to the its initial model.
- **public void refresh(ModelToView mtv):** This method refreshes view related to buff according to the current information of its model.

<u>BulletView Class inherits JavaFx ImageView</u>

Methods:

- **public void SpacecraftViewGroup(ModelToView mtv):** This constructor creates view related to bullet according to the its initial model.
- **public void refresh(ModelToView mtv):** This method refreshes view related to bullet according to the current information of its model.

<u>ModelToView Class</u>

Attributes:

- **private double locationX:** This attribute holds X location.
- **private double locationY:** This attribute holds Y location.
- **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 boolean isDead:** This attribute holds whether the object is dead in order to decide to show it on the map.

BuffModelToView Class

Attributes:

- **private BuffTypes type:** This attribute holds the type of the Buff as enum.

SpacecraftModelToView Class

Attributes:

- **private boolean isDirectionLeft:** This attribute holds the direction of the Spacecraft.
- **private boolean isMoving:** This attribute holds whether spacecraft currently moving or not.

EnemyModelToView Class

Attributes:

- **private EnemyTypes type:** This attribute holds the type of the Enemy as enum.

- **private double directionX:** This attribute holds the X direction of the enemy.
- **private double directionY:** This attribute holds the Y direction of the enemy.
- **private int health:** This attribute holds the current health point of the enemy.
- **private int maxHealth:** This attribute holds the max health point of the enemy.

<u>MeteorModelToView Class</u>

Attributes:

- **private double directionX:** This attribute holds the X direction of the Meteor object.
- **private double directionY:** This attribute holds the Y direction of the Meteor object.

<u>BulletModelToView Class</u>

Attributes:

- **private double directionX:** This attribute holds the X direction of the Bullet object.

- **private double directionY:** This attribute holds the Y direction of the Bullet object.

StationModelToView Class

Attributes:

- **private StationTypes type:** This attribute holds the type of the station as enum.
- **private int health:** This attribute holds the current health point of the station.
- **private int maxHealth:** This attribute holds the max health point of the station.

PreBossMapView Class

Attributes:

- **private double sliderLeft:** This attribute holds the beginning of the area visible on the screen.
- **private double sliderAccelerationSpeed:** This attribute holds the necessary information to provide smooth transaction of the slider.

- **private double layoutScale:** This attribute holds the scale to provide relative layout.
- **private HashMap<Long, EnemyViewGroup> enemies:** This attribute keeps track of the enemy views the map contains.
- **private HashMap<Long, EnemyStationViewGroup> stations:** This attribute keeps track of the station views in the map.
- **private HashMap<Long, BulletView> bullets:** This attribute keeps track of the bullet views in the map.
- **private HashMap<Long, BuffView> buffs:** This attribute keeps track of the buff views in the map.
- **private HashMap<Long, MeteorView> meteors:** This attribute keeps track of the meteor views in the map.
- **private SpaceCraftViewGroup spacecraft1:** This is an instance of spacecraft view class, which represents space craft of player 1.
- **private SpaceCraftViewGroup spacecraft2:** This is an instance of SpaceCraft class, which represents space craft of player 2.

Methods:

- **public void refreshEnemy(ModelToView mtv):**
- **public void refreshBullet(ModelToView mtv):**
- **public void refreshSpacecraft(ModelToView mtv):**
- **public void refreshMeteor(ModelToView mtv):**
- **public void refreshStation(ModelToView mtv):**

- **public void refreshBuff(ModelToView mtv):**

  These method above are to reflect the given parameters on the Pre Boss Map view.

RadarView Class

Attributes:

- **public HashMap<Long, ImageView> radarObjects:** This attribute holds radar objects as a view list.

Methods:

- **public void refresh(RadarObject obj):** This method refreshes the information of given parameter.

RadarObject Class

Attributes:

- **private boolean isDead:** This attribute holds whether the radar object is dead or not.
- **private long ID:** This attribute holds the specific ID of the radar object.
- **private RadarTypes type:** This attribute holds type of the radar object as enum.

- **private Location location:** This attribute holds the location of the radar object.

SpacecraftInfoView Class

Attributes:

- **private boolean isSinglePlayer:** This attribute holds the situation of the game.
- **private ImageView battery1:** This attribute holds the view of battery level of spacecraft1.
- **private ImageView battery2:** This attribute holds the view of battery level of spacecraft2.
- **private ArrayList<ImageView> smartBombs1:** This attribute holds the view of smart bomb stock of spacecraft1.
- **private ArrayList<ImageView> smartBombs2:** This attribute holds the view of smart bomb stock of spacecraft2.
- **private Rectangle healthBar1:** This attribute holds the current health percentage of spacecraft1 as view.
- **private Rectangle healthBar2:** This attribute holds the current health percentage of spacecraft2 as view.

Methods:

- **public void SpacecraftInfoView(boolean isSinglePlayer):** This method provides the initialization of this class according to given parameters.
- **public void refresh(SpacecraftInfo info):** This method refreshes the given info.

SpacecraftInfo Class

Attributes:

- **private int spacecraftNumber:** This attribute holds the number of the spacecraft (1 or 2).
- **private int health:** This attribute holds the health of the spacecraft.
- **private int maxHealth:** This attribute holds the max health level for spacecraft.
- **private int smartBombSlot:** This attribute holds the current stock of spacecraft.
- **private int battery:** This attribute holds the current battery percentage of spacecraft.

GameInfoView Class

Attributes:

- **private Label score:** This attribute holds the view of game score.

- **private Label enemyNumber:** This attribute holds the view of the enemy number.
- **private Label stationNumber:** This attribute holds the view of the station number.
- **private ImageView scoreIcon:** This attribute holds the icon for score.
- **private ImageView enemyIcon:** This attribute holds the icon for enemies.
- **private ImageView stationIcon:** This attribute holds the icon for stations.

TopBarView Class

Attributes:

- **private SpacecraftInfoView left:** This attribute holds the view for spacecrafts info.
- **private RadarView middle:** This attribute holds the view for the radar.
- **private GameInfoView right:** This attribute holds the view for the game info.

Methods:

- **public void TopBarView(boolean isSinglePlayer):** This constructor initializes the attributes according to the giving parameter.

<u>RootPane Class</u>

Attributes:

- **private boolean isSinglePlayer:** This attribute holds whether the game is single player or not.
- **private PreBossMapView preBossMapView1:** This attribute holds the map view for spacecraft1.
- **private PreBossMapView preBossMapView1:** This attribute holds the map view for spacecraft2.
- **private TopBarView topBarView:** This attribute holds the top bar view of the pre boss game.

Methods:

- **public void RootPane(boolean isSinglePlayer, double width, double height):** This constructor initializes the attributes according to the given parameters.

<u>PreBossGameController Class</u>

Attributes:

- **private RootPane rootPane:** This attribute holds the view for the Pre Boss Game as a pane.

- **private Scene scene:** This attribute holds the JavaFx scene to put the view inside.

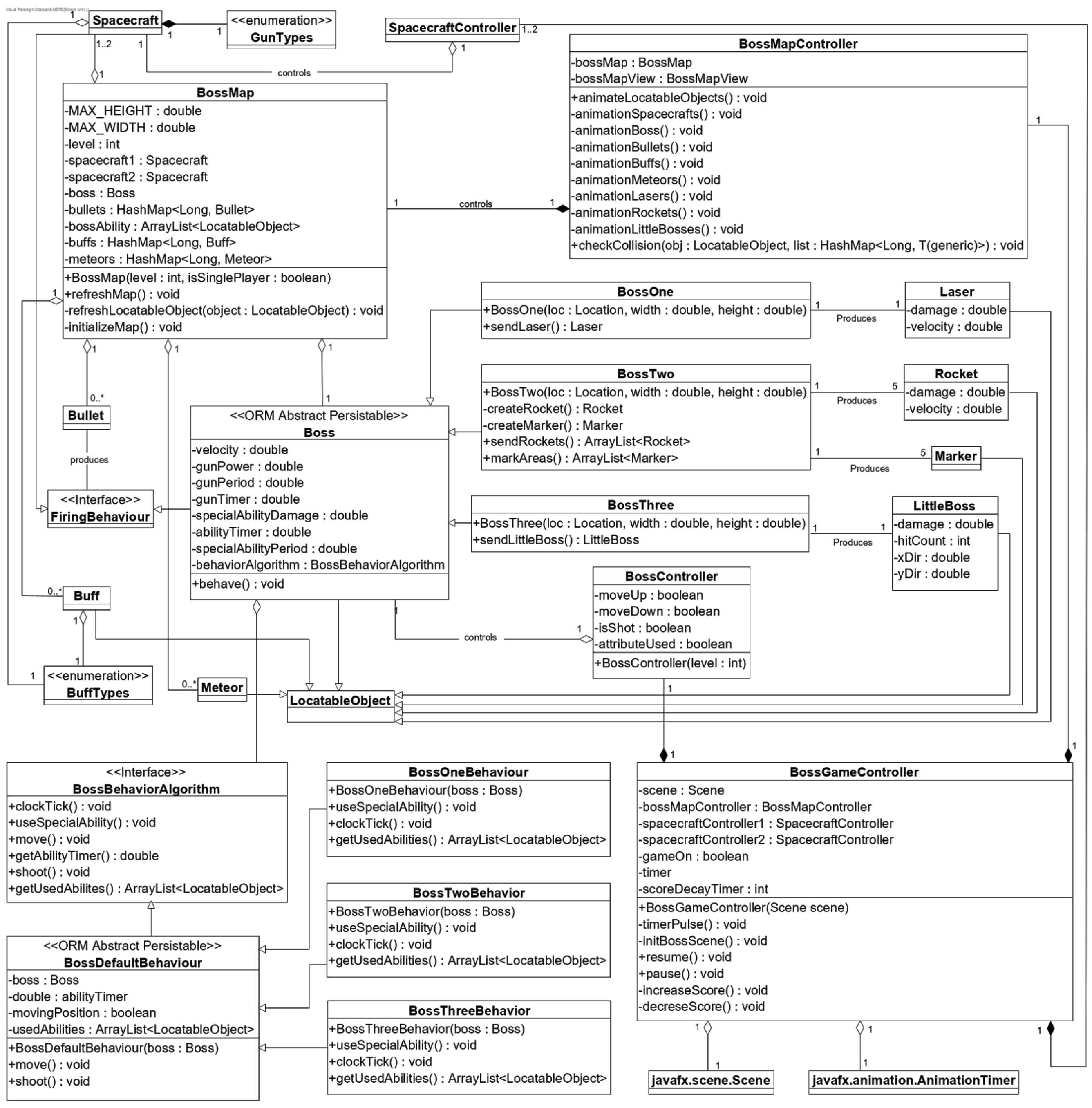
SpacecraftController Class

Attributes:

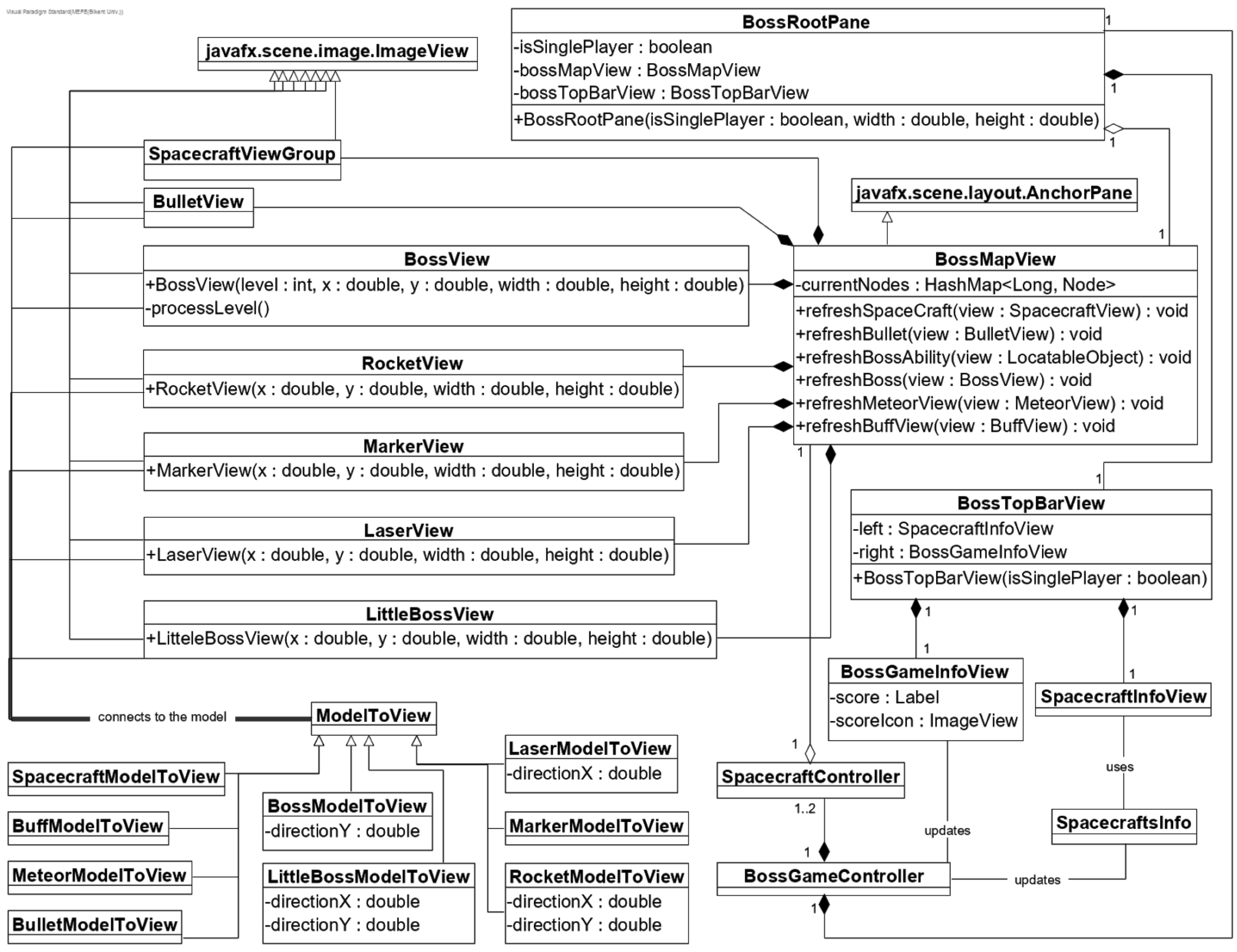
- **private PreBossMapView preBossMapView:** This attribute holds the map view for a specific spacecraft.

4.1.3. Boss Scene Class Interfaces

A-) Boss Model Management


Spacecraft
<<enumeration>> GunTypes
SpacecraftController
BossMapController
-bossMap : BossMap
-bossMapView : BossMapView
+animateLocatableObjects() : void
-animationSpacecrafts() : void
-animationBoss() : void
-animationBullets() : void
-animationBuffs() : void
-animationMeteors() : void
-animationLasers() : void
-animationRockets() : void
-animationLittleBosses() : void
+checkCollision(obj : LocatableObject, list : HashMap<Long, T(generic)>) : void
controls
BossMap
-MAX_HEIGHT : double
-MAX_WIDTH : double
-level : int
-spacecraft1 : Spacecraft
-spacecraft2 : Spacecraft
-boss : Boss
-bullets : HashMap<Long, Bullet>
-bossAbility : ArrayList<LocatableObject>
-buffs : HashMap<Long, Buff>
-meteors : HashMap<Long, Meteor>
+BossMap(level : int, isSinglePlayer : boolean)
+refreshMap() : void
-refreshLocatableObject(object : LocatableObject) : void
-initializeMap() : void
BossOne
+BossOne(loc : Location, width : double, height : double)
+sendLaser() : Laser
Laser
-damage : double
-velocity : double
Produces
BossTwo
+BossTwo(loc : Location, width : double, height : double)
-createRocket() : Rocket
-createMarker() : Marker
+sendRockets() : ArrayList<Rocket>
+markAreas() : ArrayList<Marker>
Rocket
-damage : double
-velocity : double
Marker
Bullet
produces
<<ORM Abstract Persistable>> Boss
-velocity : double
-gunPower : double
-gunPeriod : double
-gunTimer : double
-specialAbilityDamage : double
-abilityTimer : double
-specialAbilityPeriod : double
-behaviorAlgorithm : BossBehaviorAlgorithm
+behave() : void
<<Interface>> FiringBehaviour
BossThree
+BossThree(loc : Location, width : double, height : double)
+sendLittleBoss() : LittleBoss
LittleBoss
-damage : double
-hitCount : int
-xDir : double
-yDir : double
BossController
-moveUp : boolean
-moveDown : boolean
-isShot : boolean
-attributeUsed : boolean
+BossController(level : int)
Buff
<<enumeration>> BuffTypes
Meteor
LocatableObject
<<Interface>> BossBehaviorAlgorithm
+clockTick() : void
+useSpecialAbility() : void
+move() : void
+getAbilityTimer() : double
+shoot() : void
+getUsedAbilites() : ArrayList<LocatableObject>
BossOneBehaviour
+BossOneBehaviour(boss : Boss)
+useSpecialAbility() : void
+clockTick() : void
+getUsedAbilities() : ArrayList<LocatableObject>
BossTwoBehavior
+BossTwoBehavior(boss : Boss)
+useSpecialAbility() : void
+clockTick() : void
+getUsedAbilities() : ArrayList<LocatableObject>
BossThreeBehavior
+BossThreeBehavior(boss : Boss)
+useSpecialAbility() : void
+clockTick() : void
+getUsedAbilities() : ArrayList<LocatableObject>
<<ORM Abstract Persistable>> BossDefaultBehaviour
-boss : Boss
-double : abilityTimer
-movingPosition : boolean
-usedAbilities : ArrayList<LocatableObject>
+BossDefaultBehaviour(boss : Boss)
+move() : void
+shoot() : void
BossGameController
-scene : Scene
-bossMapController : BossMapController
-spacecraftController1 : SpacecraftController
-spacecraftController2 : SpacecraftController
-gameOn : boolean
-timer
-scoreDecayTimer : int
+BossGameController(Scene scene)
-timerPulse() : void
-initBossScene() : void
+resume() : void
+pause() : void
-increaseScore() : void
-decreseScore() : void
javafx.scene.Scene
javafx.animation.AnimationTimer

B) Boss View Management


Visual Paradigm Standard(METE(Bilkent Univ.))
javafx.scene.image.ImageView
BossRootPane
-isSinglePlayer : boolean
-bossMapView : BossMapView
-bossTopBarView : BossTopBarView
+BossRootPane(isSinglePlayer : boolean, width : double, height : double)
SpacecraftViewGroup
BulletView
javafx.scene.layout.AnchorPane
BossView
+BossView(level : int, x : double, y : double, width : double, height : double)
-processLevel()
BossMapView
-currentNodes : HashMap<Long, Node>
+refreshSpaceCraft(view : SpacecraftView) : void
+refreshBullet(view : BulletView) : void
+refreshBossAbility(view : LocatableObject) : void
+refreshBoss(view : BossView) : void
+refreshMeteorView(view : MeteorView) : void
+refreshBuffView(view : BuffView) : void
RocketView
+RocketView(x : double, y : double, width : double, height : double)
MarkerView
+MarkerView(x : double, y : double, width : double, height : double)
LaserView
+LaserView(x : double, y : double, width : double, height : double)
LittleBossView
+LitteleBossView(x : double, y : double, width : double, height : double)
BossTopBarView
-left : SpacecraftInfoView
-right : BossGameInfoView
+BossTopBarView(isSinglePlayer : boolean)
BossGameInfoView
-score : Label
-scoreIcon : ImageView
SpacecraftInfoView
uses
SpacecraftsInfo
connects to the model
ModelToView
SpacecraftModelToView
BuffModelToView
MeteorModelToView
BulletModelToView
BossModelToView
-directionY : double
LittleBossModelToView
-directionX : double
-directionY : double
LaserModelToView
-directionX : double
MarkerModelToView
RocketModelToView
-directionX : double
-directionY : double
SpacecraftController
1..2
BossGameController
updates
updates
1
1
1
1
1
1
1
1
1
1
1
1
1
1

Explanation of Classes

*** The following classes have been already explained in Pre Boss Class Explanation:

- LocatableObject Class
- Spacecraft Class

- SpacecraftController Class
- FiringBehaviour Interface
- Bullet Class
- Buff Class
- Meteor Class
- ScoreTable Class
- GameSituation Class
- SpacecraftViewGroup Class
- ModelToView Class
- SpacecraftModelToView Class
- BuffModelToView Class
- MeteorModelToView Class
- BulletModelToView Class
- SpacecraftInfo Class
- SpacecraftInfoView Class

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 gunPeriod:** This attribute holds the period value of Boss’s gun.
- **private double gunTimer:** This attribute holds the timer for Boss’s gun. The attribute is set to an appropriate value when the Boss shoots.
- **private double specialAbilityDamage:** This attribute holds the damage of Boss’s special ability.
- **private double abilityTimer:** This attribute holds the timer for special ability of the Boss. The attribute is set to an appropriate value when the Boss uses its special ability.
- **private double specialAbilityPeriod:** This method holds the period value for Boss’s special ability.
- **private BossBehaviourAlgorithm behaviourAlgorithm:** This attribute holds the algorithm that will be used for the Boss. The method is created as a result of strategy design pattern.

Methods:

- **public void behave():** This method makes the Boss act by given BossBehaviourAlgorithm.

<u>BossBehaviourAlgorithm Interface</u>

Methods:

- **public void clockTick():** This method causes to decrease of ability timer.
- **public void useSpecialAbility():** The method makes the Boss use its special ability.
- **public void move():** The method makes the Boss move.
- **public double shoot():** The method makes the Boss shoot.

BossDefaultBehaviour Abstract Class

Attributes:

- **private Boss boss:** This attribute holds a representation of a Boss.
- **private double abilityTimer:** This attribute holds the abilityTimer of the Boss.
- **private boolean movingPosition:** The attribute is used to determine the direction of the Boss.
- **private ArrayList<LocatableObject> usedAbilities:** Abilities created by the Boss is hold here.

Methods:

- **public BossDefaultBehaviour(Boss boss):** This is constructor of BossDefaultBehaviour which takes a Boss reference as parameter.
- **public void move():** This method provides Boss to move up or down.

- **public void shoot():** This method provides Boss to fire bullets.

BossOneBehavior Class

Methods:

- **public BossOneBehaviour(Boss boss):** This is the constructor of BossOneBehaviour which takes a Boss reference.
- **public void useSpecialAbility():** This method provides BossOne to use its special ability (laser).
- **public void clockTick():** Ability timer of the Boss1 is decreased according to the method.

BossTwoBehaviour Class

BossTwo behaves according to this class.

Because of the strategy design pattern, the methods of the class have same functionality with BossOneBehaviour. The only difference is implementation.

BossThreeBehaviour Class

BossThree behaves according to this class.

See BossTwoBehaviour Class to get information about methods.

BossOne Class

Methods:

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

BossTwo Class

Methods:

- **public BossTwo(Location location, double width, double height):** This is constructor of BossTwo who takes location, width and height as parameter.
- **public ArrayList<Rocket> sendRockets():** Creates 5 rockets to send.
- **public ArrayList <Marker> markAreas():** Creates 5 locations to be marked.
- **private Rocket createRocket():** It creates a rocket to help sendRockets() method.

- **private Marker createMarker():** It creates a marker to help markAreas() method.

BossThree Class

Methods:

- **public BossThree(Location location, double width, double height):** This is constructor of BossThree which takes location, width and height as parameter.
- **public LittleBoss sendLittleBoss():** This method provides BossThree 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.
- **private double velocity:** This attribute holds the velocity of laser.

Rocket Class

Attributes:

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

- **private double velocity**: The attribute holds the velocity of a rocket.

Marker Class

This class provides the BossTwo to mark areas which it sends rocket as a warning.

LittleBoss Class

Attributes:

- **private double damage:** This attribute holds the damage given by Little Boss.
- **private double hitCount:** Because the objects of this class will be destroyed after the objects hit 3 times to boundaries of the BossMap, the attribute holds the value of how much left.
- **private double xDir:** The attribute holds the x direction, which LittleBoss is heading to.
- **private double yDir:** The attribute holds the y direction, which LittleBoss is heading to.
- **private double velocity:** The attribute holds the velocity of the LittleBoss.

BossController Class

Attributes:

- **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 spacecraft1:** This is an instance of Spacecraft class, which represents the player's spacecraft.
- **private Spacecraft spacecraft2:** This is an instance of Spacecraft class, which represents the second player's spacecraft if it is two-players game mood.
- **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 ArrayList <LocatableObject> bossAbility:**This attribute represents the special abilities that are created by the Boss.
- **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.

Methods:

- **public BossMap(int level):** This is the constructor for the BossMap. Also, the Boss of the map will be created according to level value.
- **public void refreshMap():** This method organizes map again for changes. This method should be called continuously except pause.

- **private void refreshLocatableObject(LocatableObject object):** This method refreshes object in the game for changes. This method should be called continuously except pause.
- **private void initializeMap():** This method creates map according to level at the beginning of the Boss scene.

BossMapController Class

Attributes:

- **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.

Methods:

- **public void animateLocatableObject():** This method checks changes in every locatable object in the map.
- **private void animationSpacecrafts():** This method checks changes in spacecrafts.
- **private void animationBoss():** This method checks changes in boss.

- **private void animationBullets():** This method checks changes in bullets.
- **private void animationBuffs():** This method checks changes in buffs.
- **private void animationMeteors():** This method checks changes in meteors.
- **private void animationLasers():** This method checks changes in lasers.
- **private void animationRockets():** This method checks changes in rockets.
- **private void animationLittleBosses():** This method checks changes in little bosses.
- **public void checkCollisons():** This methods controls collision of two objects.

BossGameController Class

Attributes:

- **private Scene scene:** This attribute holds the JavaFx scene to put the view inside.
- **private SpacecraftContoller spacecraftController1:** This attribute holds an instance of SpacecraftController to check the player's inputs.
- **private SpacecraftContoller spacecraftController2:** This attribute holds an instance of SpacecraftController to check the second player's inputs if it is two-players game mood.

- **private boolean gameOn:** This attribute holds if the game continues or it stops.
- **private javafx.animation.AnimationTimer timer:** This attribute is used to create time pulses for continuity of the game.
- **private int scoreDecayTimer:** This attribute is to provide a timer for score decay logic.

Methods:

- **public BossGameController(Scene scene):** This is a constuctor of BossGameController which takes scene as a parameter.
- **private void timerPulse():** This method is called at every time pulse created by AnimationTimer and
- **private void initBossScene():** This method initializes BossMap according to parameters in the constructor.
- **public void resume():** This method will resume the paused game.
- **public void pause():** This method will pause the resuming game.
- **private void increaseScore():** This method will increase the current score with the given amount.
- **private void decreaseScore():** This method will decrease the current score with the given amount.

BossMapView Class

The class extends javafx.scene.layout.AnchorPane to put and manage images in the layout.

Attributes:

- **private HashMap <Long, Node> currentNodes:** The attribute holds the image views of all created images.

Methods:

- **public void refreshSpacecraftView (SpacecraftView spacecraftView):** The method refreshes the image of the spacecraft.
- **public void refreshBulletView(BulletView bulletView):** The method refreshes the image of the given BulletView.
- **public void refreshBossView(BossView bossView):** The method refreshes the image of the BossView.
- **public void refreshBossAbility(LocatableObject view):** The method refreshes the image of the Boss ability according to given ability.
- **public void refreshBuffView(BuffView buffView):** The method refreshes the image of given BuffView.
- **public void refreshMeteorView(MeteorView meteorView):** The method refreshes the image of given MeteorView.

BossView Clas

The class extends javafx.scene.image.ImageView to represent an image.

Methods:

- **public BossView(int level, double x, double y, double width, double height):** View of the Boss is created according to given parameters
- **private void processLevel(int level):** View of the Boss is selected according to given level value. The method helps to find appropriate BossView.

LaserView Class

Methods:

- **public LaserView(double x, double y, double width, double height):** View of the Laser is created according to given parameters.

RocketView Class

Methods:

- **public RocketView(double x, double y, double width, double height):** View of the Rockets are created according to given parameters.

MarkerView Class

Methods:

- **public MarkerView(double x, double y, double width, double height):** View of the Markers are created according to given parameters.

LittleBossView Class

Methods:

- **public LittleBossView (double x, double y, double width, double height):** View of the LittleBosses are created according to given parameters.

BossModelToView Class

Attributes:

- **private double directionY:** This attribute holds y direction of the boss.

LaserModelToView Class

Attributes:

- **private double directionX:** This attribute holds x direction of the laser.

RocketModelToView Class

Attributes:

- **private double directionX:** This attribute holds x direction of the rocket.
- **private double directionY:** This attribute holds y direction of the rocket.

LittleBossModelToView Class

Attributes:

- **private double directionX:** This attribute holds x direction of the little boss.
- **private double directionY:** This attribute holds y direction of the little boss.

BossGameInfoView Class

Attributes:

- **private Label score:** This attribute holds the score of the player.

- **private ImageView scoreIcon:** This attribute holds the icon which represent the score as an ImageView.

BossTopBarView Class

Attributes:

- **private SpacecraftInfoView left:** This is an instance of SpacecraftInfoView to be replaced the left side of the top.
- **private BossGameInfoView right:** This is an instance of BossGameInfoView to be replaced the right side of the top.

Methods:

- **public BossTopBarView(boolean isSinglePlayer):** This is a constructor of BossTopBarView which takes a parameter which shows the game is single player or not.

BossRootPane Class

Attributes:

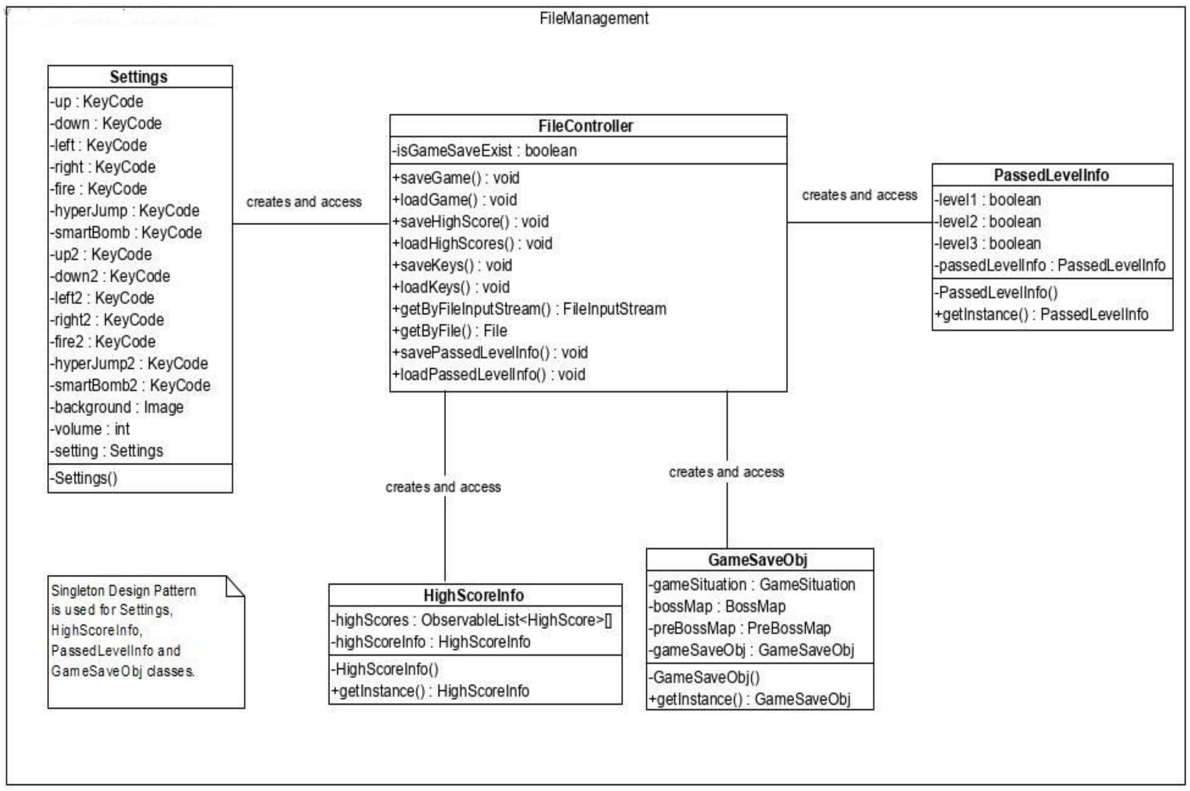
- **private boolean isSinglePlayer:** This attribute shows whether the game is single player or not.
- **private BossMapView bossMapView:** This is an instance of BossMapView to show the game on the screen.

- **private BossTopBarView bossTopBarView:** This is an instance of BossTopBarView.

Methods:

- **public BossRootPane(boolean isSinglePlayer, double width, double height):** This is a constructor of BossRootPane which takes width and height of the screen. It also takes another parameter which shows the game is single player or not.

4.1.4. File Management Class Interfaces



*** The following classes have been already explained in section 4.1.1.3.

- ➔ GameSaveObj Class
- ➔ PassedLevelInfo Class
- ➔ Settings Class
- ➔ HighScoreInfo Class

FileController Class

Attributes:

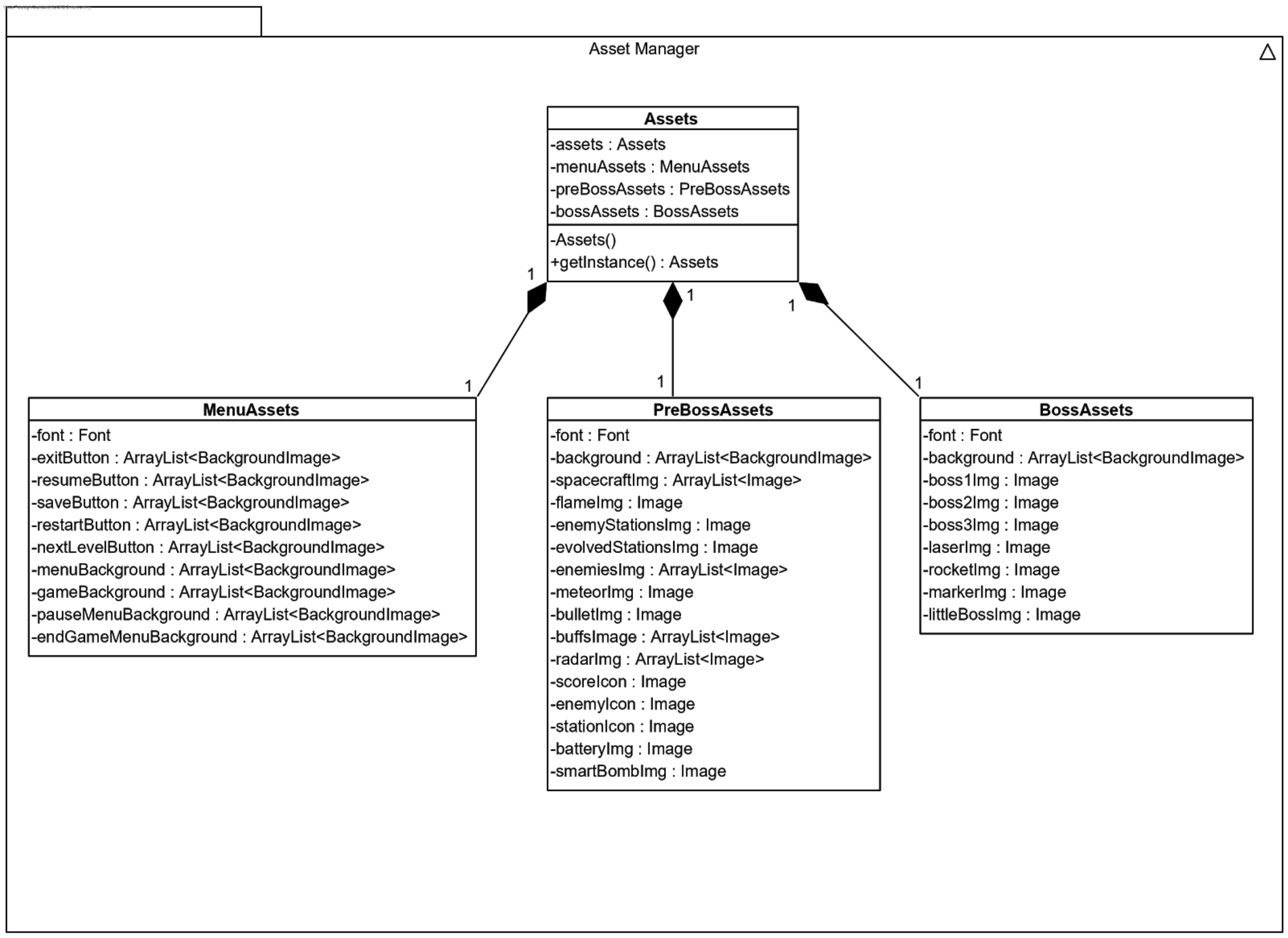
- **private boolean isGameSaveExist:** This attribute is set true when a game is recorded to the files.

Methods:

- **public void saveGame() :** This method records gameSaveObj into the file.
- **public void loadGame() :** This method loads game objects from the file.
- **public void saveHighScore() :** This method records highScoreInfo to the file.
- **public void loadHighScores() :** This method loads high scores from the file.
- **public void saveKeys() :** This method records settings to the file.
- **public void loadKeys() :** This method loads setting information from the file.

- **public FileInputStream getByFileInputStream() :** This method is needed to get data using file input stream.
- **public File getByFile() :** This method is needed to get data using file.
- **public void savePassedLevelInfo() :** This method records passedLevelInfo to the file.
- **public void loadPassedLevelInfo() :** This method loads passed level information from the file.

4.1.5. Asset Management Class Interfaces



Asset Manager
Assets
-assets : Assets
-menuAssets : MenuAssets
-preBossAssets : PreBossAssets
-bossAssets : BossAssets
-Assets()
+getInstance() : Assets
1
1
1
1
1
1
MenuAssets
-font : Font
-exitButton : ArrayList<BackgroundImage>
-resumeButton : ArrayList<BackgroundImage>
-saveButton : ArrayList<BackgroundImage>
-restartButton : ArrayList<BackgroundImage>
-nextLevelButton : ArrayList<BackgroundImage>
-menuBackground : ArrayList<BackgroundImage>
-gameBackground : ArrayList<BackgroundImage>
-pauseMenuBackground : ArrayList<BackgroundImage>
-endGameMenuBackground : ArrayList<BackgroundImage>
PreBossAssets
-font : Font
-background : ArrayList<BackgroundImage>
-spacecraftImg : ArrayList<Image>
-flameImg : Image
-enemyStationsImg : Image
-evolvedStationsImg : Image
-enemiesImg : ArrayList<Image>
-meteorImg : Image
-bulletImg : Image
-buffsImage : ArrayList<Image>
-radarImg : ArrayList<Image>
-scoreIcon : Image
-enemyIcon : Image
-stationIcon : Image
-batteryImg : Image
-smartBombImg : Image
BossAssets
-font : Font
-background : ArrayList<BackgroundImage>
-boss1Img : Image
-boss2Img : Image
-boss3Img : Image
-laserImg : Image
-rocketImg : Image
-markerImg : Image
-littleBossImg : Image

MenuAssets

Attributes:

- **private Font font:** This attribute holds the font that will be used in menu.

- **private ArrayList<BackgroundImage> exitButton:** This attribute holds the background images of different themes for exit button.
- **private ArrayList<BackgroundImage> resumeButton:** This attribute holds the background images of different themes for resume button.
- **private ArrayList<BackgroundImage> saveButton:** This attribute holds the background images of different themes for save button.
- **private ArrayList<BackgroundImage> restartButton:** This attribute holds the background images of different themes for restart button.
- **private ArrayList<BackgroundImage> nextLevelButton:** This attribute holds the background images of different themes for next level button.
- **private ArrayList<BackgroundImage> menuBackground:** This attribute holds the background images of different themes for menu stage.
- **private ArrayList<BackgroundImage> gameBackground:** This attribute holds the background images of different themes for game stage.
- **private ArrayList<BackgroundImage> pauseMenuBackground:** This attribute holds the background images of different themes for pause menu stage.
- **private ArrayList<BackgroundImage> endGameMenuBackground:** This attribute holds the background images of different themes for end game menu stage.

PreBossAssets

Attributes:

- **private Font font:** This attribute holds the font that will be used in pre boss game stage.
- **private ArrayList<BackgroundImage> background:** This attribute holds the background of different themes that will be used in pre boss game stage.
- **private ArrayList<Image> spacecraftImg:** This attribute holds the different images for spacecraft that will be used in game stage.
- **private Image flameImg:** This attribute holds the image for flame of spacecraft that will be used in game stage.
- **private Image enemyStationsImg:** This attribute holds the image for enemy stations that will be used in game stage.
- **private Image evolvedStationsImg:** This attribute holds the image for evolved enemy stations that will be used in game stage.
- **private ArrayList<Image> enemiesImg:** This attribute holds the different images for enemies that will be used in game stage.
- **private Image meteorImg:** This attribute holds the image for meteor that will be used in game stage.
- **private Image bulletImg:** This attribute holds the image for bullet that will be used in game stage.

- **private ArrayList<Image> buffsImg:** This attribute holds the different images for buffs that will be used in game stage.
- **private ArrayList<Image> radarImg:** This attribute holds the different images for radar objects that will be used in game stage.
- **private Image scoreIcon:** This attribute holds the icon for score that will be used in game stage.
- **private Image enemyIcon:** This attribute holds the icon for enemy that will be used in game stage.
- **private Image stationIcon:** This attribute holds the icon for stations that will be used in game stage.
- **private Image batteryImg:** This attribute holds the image for battery that will be used in game stage.
- **private Image smartBombImg:** This attribute holds the image for smart bomb that will be used in game stage.

BossAssets

Attributes:

- **private Font font:** This attribute holds the font that will be used in boss game stage.
- **private ArrayList<BackgroundImage> background:** This attribute holds the background of different themes that will be used in boss game stage.

- **private Image boss1Img:** This attribute holds the image for BossOne that will be used in game stage.
- **private Image boss2Img:** This attribute holds the image for BossTwo that will be used in game stage.
- **private Image boss3Img:** This attribute holds the image for BossThree that will be used in game stage.
- **private Image laserImg:** This attribute holds the image for laser that will be used in game stage.
- **private Image rocketImg:** This attribute holds the image for rocket that will be used in game stage.
- **private Image markerImg:** This attribute holds the image for marker that will be used in game stage.
- **private Image littleBossImg:** This attribute holds the image for little boss that will be used in game stage.

Assets Class

Attributes:

- **private Assets assets:** This attribute holds an instance of this class to provide singleton.
- **private MenuAssets menuAssets:** This attribute holds the menu assets.
- **private PreBossAssets preBossAssets:** This attribute holds the pre boss assets.

- **private BossAssets bossAssets:** This attribute holds the boss assets.

Methods:

- **private void Assets():** This constructor is private to provide singleton mechanism.
- **public Assets getInstance():** This method provide the same singleton instance of this class.

4.2. Design Decisions and Trade-offs

4.2.1. Pre Boss Game Design Decisions

- We decided to apply Strategy Design Pattern to the firing behavior in the Pre Boss Game scene since although firing pattern can be slightly different most of the attributes and implementation is similar. By applying this pattern, we avoid a partial repetition. Although the benefit is not dramatic at this level; in the future if we decide to extend the game, this could save a lot of time and make it easy to maintain the software.
- We decided to apply Factory Design Pattern to provide enemy production in a wiser manner. There are two classes that produces enemy which are EnemyStation and EvolvedEnemyStation. By creating a enemy factory we avoid from duplicate code during implementation and possibly in the future if we decide to add new classes with the same future.
- We decided **not** to apply Observer Design Pattern when we update view classes according to the map model. At the beginning, we thought that it could be applied; however, since the map model is constantly changing view classes would have to be changed too much that it would decrease our game performance. Rather than applying Observer Design Pattern, we decided to update the view classes at specific time intervals by the help of the timer in the PreBossGameController class.

4.2.2 Boss Game Design Decisions

- We decided to apply Strategy Design Pattern to the behaviours of the Bosses. The reason is bosses can shoot differently or move differently or use different special abilities. By applying the pattern, we ensure that the code stays cleaner and organized. It also helps to change the code easily, when additional bosses are added.
- We decided to apply Singleton Design Pattern to BossMap class because there will be only one BossMap object through the game. We want to make sure that this requirement is not violated.

4.2.3 Assets Manager Design Decisions

- We decided to apply Singleton Design Pattern to Assets class. Since all of the assets that will be used in the game have to be brought from the file, we don’t need to do this process over and over again in game. Just at the initialization of the software we can bring them from the file and keep them in a class. In the game when any asset is needed, we can get an instance of this class and this instance can be the same instance for everyone. Therefore, we applied Singleton to Assets class. One good benefit of this is to separate concerns. In game classes we don’t have to worry about how to get the assets from file.

4.2.4 Menu Design Decisions

- We decided to apply Singleton design pattern to model classes of Menu Subsystem. These classes are used by many classes to get information and they are supposed to have only one instance throughout the game. Therefore, by making them singleton, we are sure that the classes get correct information.

4.3. Packages

4.3.1. External Packages

javafx.application: Application class in the package is used to start a working javafx project.

javafx.stage: Stage class in the package is used to provide a container to hold the nodes that will be created by the developers.

javafx.scene.image: Image and ImageView classes are used to load and display images. Mentioned classes are provided in the package.

javafx.scene.layout: Pane class is used to put images in correct positions and manipulate them. Background and BackgroundImage classes are used to create a background. All mentioned classes are found in the package.

javafx.animation: AnimationTimer class provided by the package is used to update all the views for every frame.

java.io: FileInputStream class in the package is used to find the directory of the images.

java.util: The package provides HashMap and ArrayList classes which are used to keep track of objects.

4.3.2. Internal Packages

controller.assetsController: The package includes classes which give desired game assets to caller class. For example, BossAssets class provides Boss images.

controller.fileController: The package includes FileController Class which reads necessary files. For example, it can read highest scores or saved game keys.

controller.menuController: The package includes classes that are responsible for passing menu scene to game scenes.

controller.preBossGameController: Classes related to business of Pre Boss Game are hold here. The package includes PreBossGameController and PreBossMapController classes.

controller.bossGameController: Classes related to business of Boss Game are hold here. The package includes BossGameController, BossMapController and BossController classes.

model.entities: The package includes all entities like Spacecraft, Enemy, Boss and Bullet classes.

model.PreBossMap: The package provides the container named PreBossMap class.

model.BossMap: The package provides the container named BossMap.

utilization: The package provides classes to facilitate conversion between model classes and view classes. The package includes classes like ModelToView and ModelToViewSpacecraft.

view.preBossMapView: The package provides the container named preBossMapView.

view.BossMapView: The package provides the container named BossMapView.

5. Improvement summary

After Design Report Iteration 1, we have made following changes:

- Attempts are made to correct grammatical mistakes.
- Design goals and design trade-offs are changed. We have tried to add reasonable design goals.
- Subsystem decomposition is changed.

- Classes for File Management, Menu Management and Asset Management are added in low level design.
- Model and Controller classes are updated and View classes are added for Pre Boss Game.
- Model and Controller classes are updated and View classes are added for Boss Game.
- We added classes related to File and Asset management.
- External and internal packages are added.
- We applied design patterns to our classes.