CS 319 - Object-Oriented Software Engineering

System Design Report

Galaxy Invader

<u>Group 3J</u>

Umer Shamaan

Osman Can Yıldız

Denizhan Yağcı

Cemre Osan

Tanay Toksoy

Table of Contents

1. Introduction

Galaxy Invaders is a 2D side-scrolling and shooting game. The player can do moving and shooting actions in order to reach the purposes of the game.

The System Design Document represents the design plan which is necessary for the implementation part. The design plan described, follows the requirements specified in the Project Analysis Report prepared.

1.1. Purpose of the System Design Document (SDD)

The system design document (SDD) consists of detailed steps of design process. This document describes the system architecture, subsystem services and low-level design. It provides a foundational guide for further implementation details all the way to an executable solution. [1]

1.2. Design Goals

Galaxy Invaders design made according to following concepts

1.2.1 Trade-Offs

1.2.1.1. User-Friendliness

Galaxy Invaders will have a user friendly interface that any player who is playing for the first time have no problem with the menus or the controllers. The menu will offer an option called “How to Play” which introduces the controls to the player.

1.2.1.2. Performance

Galaxy Invaders will be a fast paced game. Hence one of the main goals is making the system as fast as possible and reducing the waiting time for the player between levels.

1.2.1.3. Maintainability

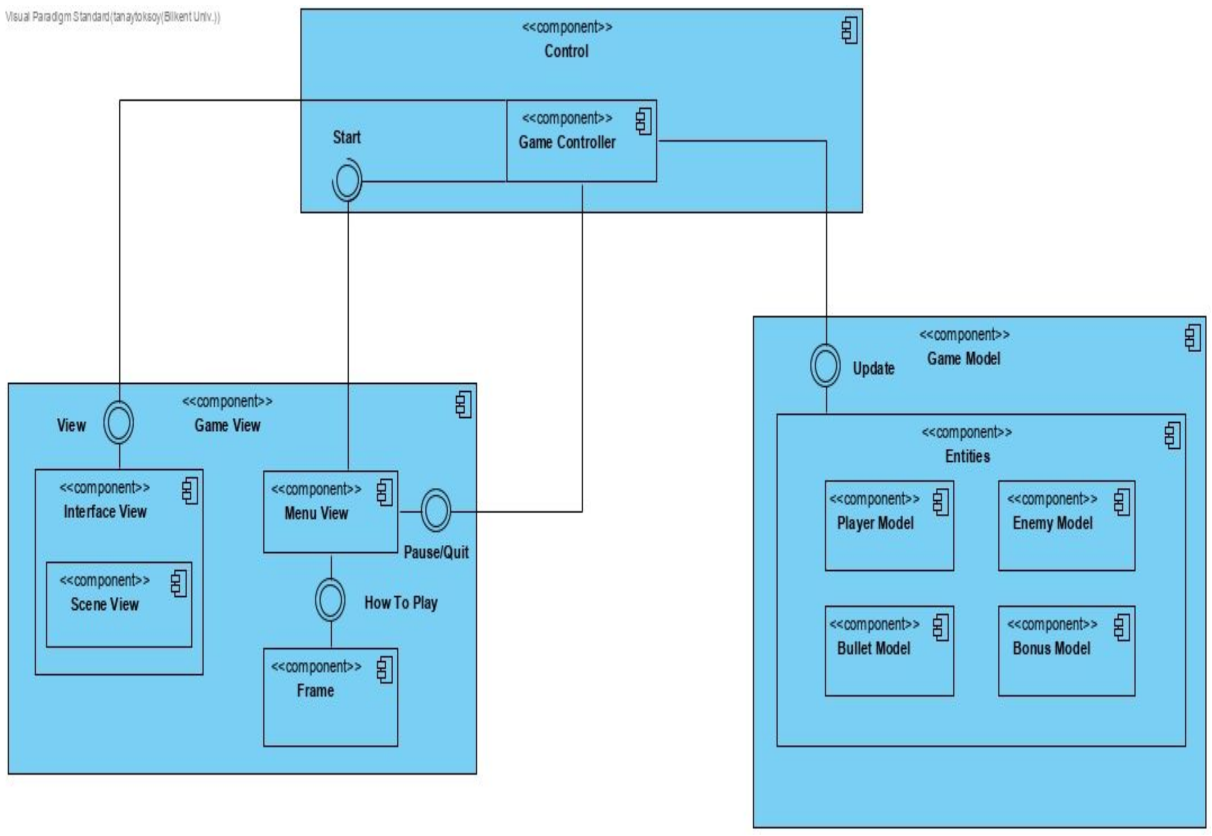
Galaxy Invaders will be based on object oriented programming concept. Thus it will be easy to add new features and change already existed features.

1.2.1.4. Robustness

In order for the player to enjoy the game, Galaxy Invaders should have minimum bugs and the programs unintended influence should be minimized. Therefore users influence on overall game will be limited however this will not affect user experience with limited actions or controls.

2. System Architecture

2.1. Subsystem Decomposition



2.2. Hardware/Software Mapping

Since this game is going to be implemented using java language, it is going to need java runtime environment. The computation rate will not be too demanding so there is no need for multiple processors, even a low-end processor will be enough to run the game. The user will need a keyboard to play the game. All of the movements of the spaceship will be handled with w, a, s, d keys to move up, left, down, right respectively. The high scores of the game will be stored in a text file which means that there is no need for an internet connection.

2.3. Persistent Data Management

The only thing that needs to be stored is high scores so there is no need to use a cloud or a database, thus they will be stored in a text file.

2.4. Access Control and Security

Every user in this game has the same rights and there is no login mechanism or any other type of authentication which means that there is no need to put up any access control and security measurements.

2.5. Boundary Conditions

As the app will not have an .exe file there won't be any need to install the game. However, it will have an executable jar file. This will make the transfer of game from one platform to another fairly simple.

The two ways to exit from the app are to click on the **Quit** button on the main menu are to click on the exit button on the game window.

If the execution of the game is abruptly stopped during gameplay, the current score data will be lost.

The application might also crash due to corrupt music files or text files. Simply replacing those files with the correct ones will fix the issue.

3. Subsystem Services

3.1. Control Subsystem

The control system will include the game controller. Game controller's job is to communicate with both the Game View and Game Model subsystems and update them.

3.2. Game View Subsystem

This subsystem will include interface and menu view. It’s job is to display the relevant components depending on the data it gets from control subsystem.

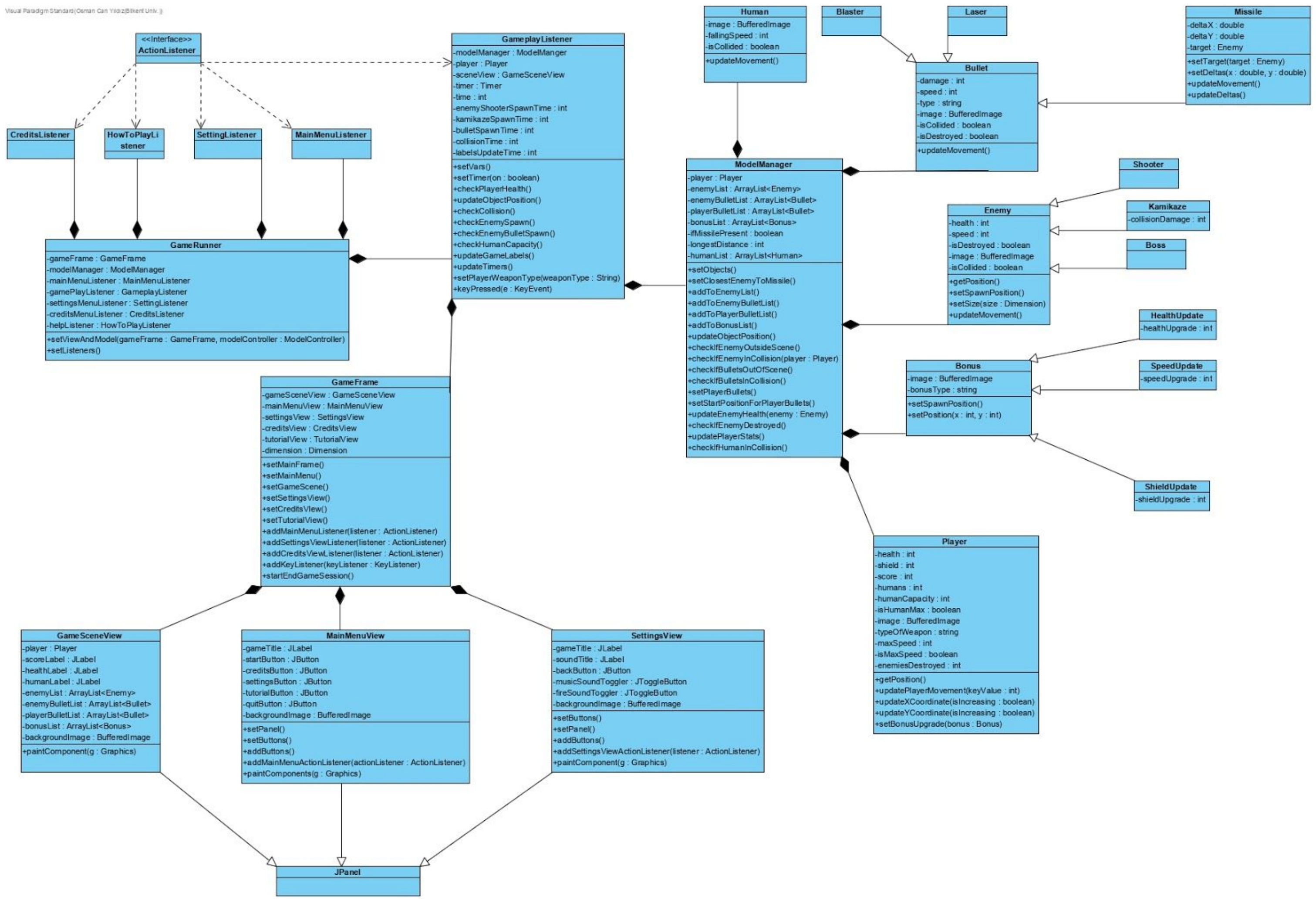
3.3. Game Model Subsystem

This Subsystem will include all of the game models which are the player, enemies, bullets and bonus. It’s job is to modify the models depending on the control subsystem.

4. Low-level Design

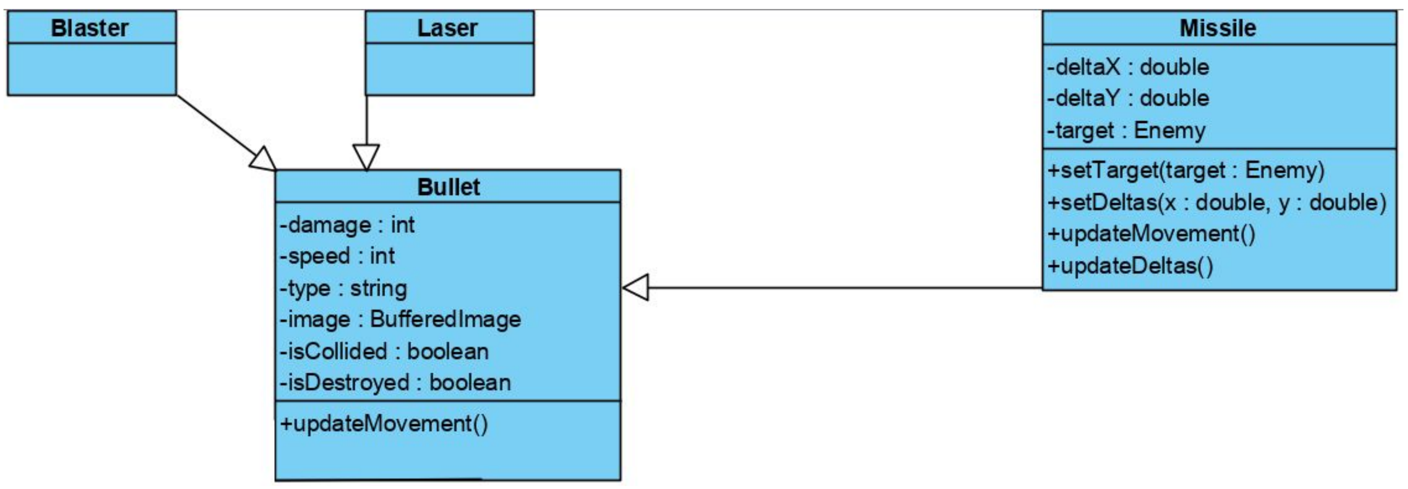
4.1. Final Object Design

Final Object Design is completed as seen below. It includes models, interfaces and classes.

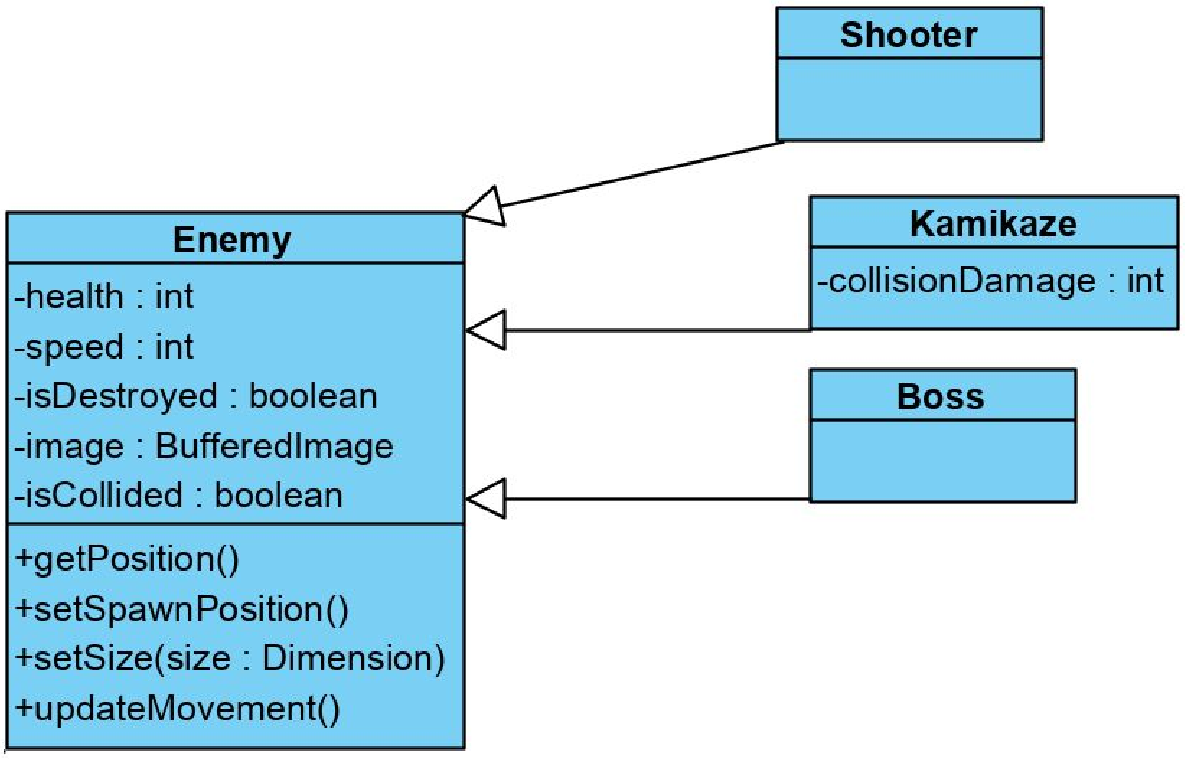


Visual Paradigm Standard(Osman Can Yildiz(Bilkent Univ.))
<<Interface>>
ActionListener
CreditsListener
HowToPlayListener
SettingListener
MainMenuListener
GameplayListener
-modelManager : ModelManger
-player : Player
-sceneView : GameSceneView
-timer : Timer
-time : int
-enemyShooterSpawnTime : int
-kamikazeSpawnTime : int
-bulletSpawnTime : int
-collisionTime : int
-labelsUpdateTime : int
+setVars()
+setTimer(on : boolean)
+checkPlayerHealth()
+updateObjectPosition()
+checkCollision()
+checkEnemySpawn()
+checkEnemyBulletSpawn()
+checkHumanCapacity()
+updateGameLabels()
+updateTimers()
+setPlayerWeaponType(weaponType : String)
+keyPressed(e : KeyEvent)
GameRunner
-gameFrame : GameFrame
-modelManager : ModelManager
-mainMenuListener : MainMenuListener
-gamePlayListener : GameplayListener
-settingsMenuListener : SettingListener
-creditsMenuListener : CreditsListener
-helpListener : HowToPlayListener
+setViewAndModel(gameFrame : GameFrame, modelController : ModelController)
+setListeners()
GameFrame
-gameSceneView : GameSceneView
-mainMenuView : MainMenuView
-settingsView : SettingsView
-creditsView : CreditsView
-tutorialView : TutorialView
-dimension : Dimension
+setMainFrame()
+setMainMenu()
+setGameScene()
+setSettingsView()
+setCreditsVIew()
+setTutorialView()
+addMainMenuListener(listener : ActionListener)
+addSettingsViewListener(listener : ActionListener)
+addCreditsViewListener(listener : ActionListener)
+addKeyListener(keyListener : KeyListener)
+startEndGameSession()
GameSceneView
-player : Player
-scoreLabel : JLabel
-healthLabel : JLabel
-humanLabel : JLabel
-enemyList : ArrayList<Enemy>
-enemyBulletList : ArrayList<Bullet>
-playerBulletList : ArrayList<Bullet>
-bonusList : ArrayList<Bonus>
-backgroundImage : BufferedImage
+paintComponent(g : Graphics)
MainMenuView
-gameTitle : JLabel
-startButton : JButton
-creditsButton : JButton
-settingsButton : JButton
-tutorialButton : JButton
-quitButton : JButton
-backgroundImage : BufferedImage
+setPanel()
+setButtons()
+addButtons()
+addMainMenuActionListener(actionListener : ActionListener)
+paintComponents(g : Graphics)
SettingsView
-gameTitle : JLabel
-soundTitle : JLabel
-backButton : JButton
-musicSoundToggler : JToggleButton
-fireSoundToggler : JToggleButton
-backgroundImage : BufferedImage
+setButtons()
+setPanel()
+addButtons()
+addSettingsViewActionListener(listener : ActionListener)
+paintComponent(g : Graphics)
JPanel
Human
-image : BufferedImage
-fallingSpeed : int
-isCollided : boolean
+updateMovement()
Blaster
Laser
Missile
-deltaX : double
-deltaY : double
-target : Enemy
+setTarget(target : Enemy)
+setDeltas(x : double, y : double)
+updateMovement()
+updateDeltas()
Bullet
-damage : int
-speed : int
-type : string
-image : BufferedImage
-isCollided : boolean
-isDestroyed : boolean
+updateMovement()
ModelManager
-player : Player
-enemyList : ArrayList<Enemy>
-enemyBulletList : ArrayList<Bullet>
-playerBulletList : ArrayList<Bullet>
-bonusList : ArrayList<Bonus>
-ifMissilePresent : boolean
-longestDistance : int
-humanList : ArrayList<Human>
+setObjects()
+setClosestEnemyToMissile()
+addToEnemyList()
+addToEnemyBulletList()
+addToPlayerBulletList()
+addToBonusList()
+updateObjectPosition()
+checkIfEnemyOutsideScene()
+checkIfEnemyInCollision(player : Player)
+checkIfBulletsOutOfScene()
+checkIfBulletsInCollision()
+setPlayerBullets()
+setStartPositionForPlayerBullets()
+updateEnemyHealth(enemy : Enemy)
+checkIfEnemyDestroyed()
+updatePlayerStats()
+checkIfHumanInCollision()
Shooter
Enemy
-health : int
-speed : int
-isDestroyed : boolean
-image : BufferedImage
-isCollided : boolean
+getPosition()
+setSpawnPosition()
+setSize(size : Dimension)
+updateMovement()
Kamikaze
-collisionDamage : int
Boss
HealthUpdate
-healthUpgrade : int
Bonus
-image : BufferedImage
-bonusType : string
+setSpawnPosition()
+setPosition(x : int, y : int)
SpeedUpdate
-speedUpgrade : int
ShieldUpdate
-shieldUpgrade : int
Player
-health : int
-shield : int
-score : int
-humans : int
-humanCapacity : int
-isHumanMax : boolean
-image : BufferedImage
-typeOfWeapon : string
-maxSpeed : int
-isMaxSpeed : boolean
-enemiesDestroyed : int
+getPosition()
+updatePlayerMovement(keyValue : int)
+updateXCoordinate(isIncreasing : boolean)
+updateYCoordinate(isIncreasing : boolean)
+setBonusUpgrade(bonus : Bonus)

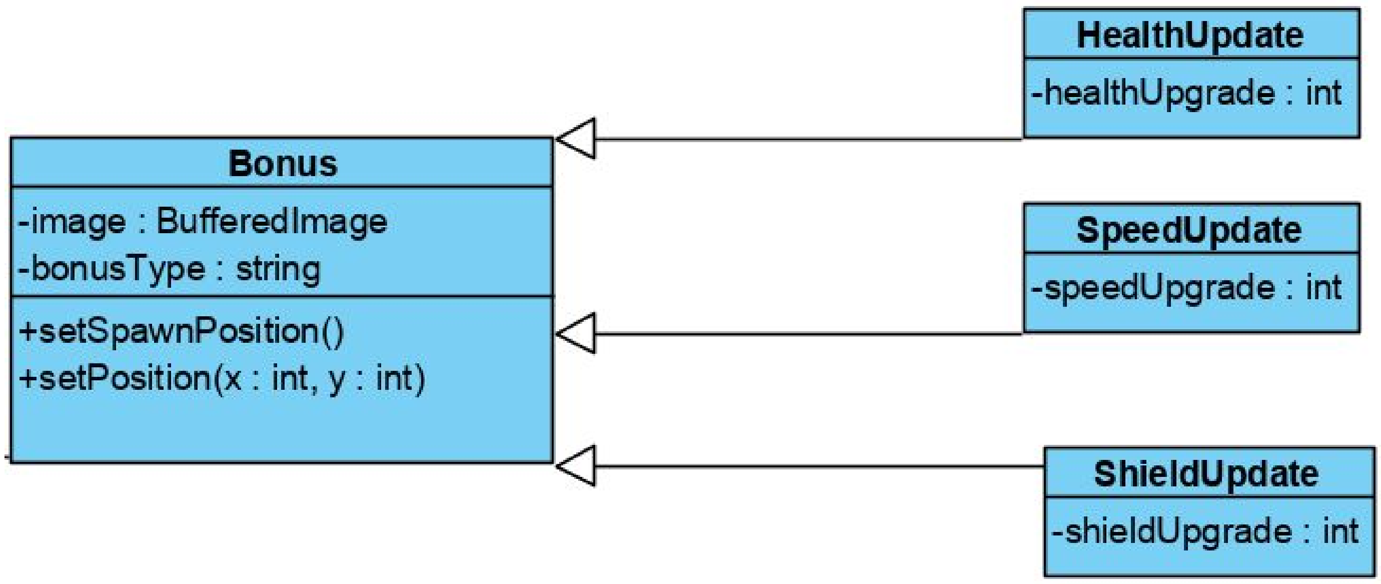
Piece 1: Bullet Design



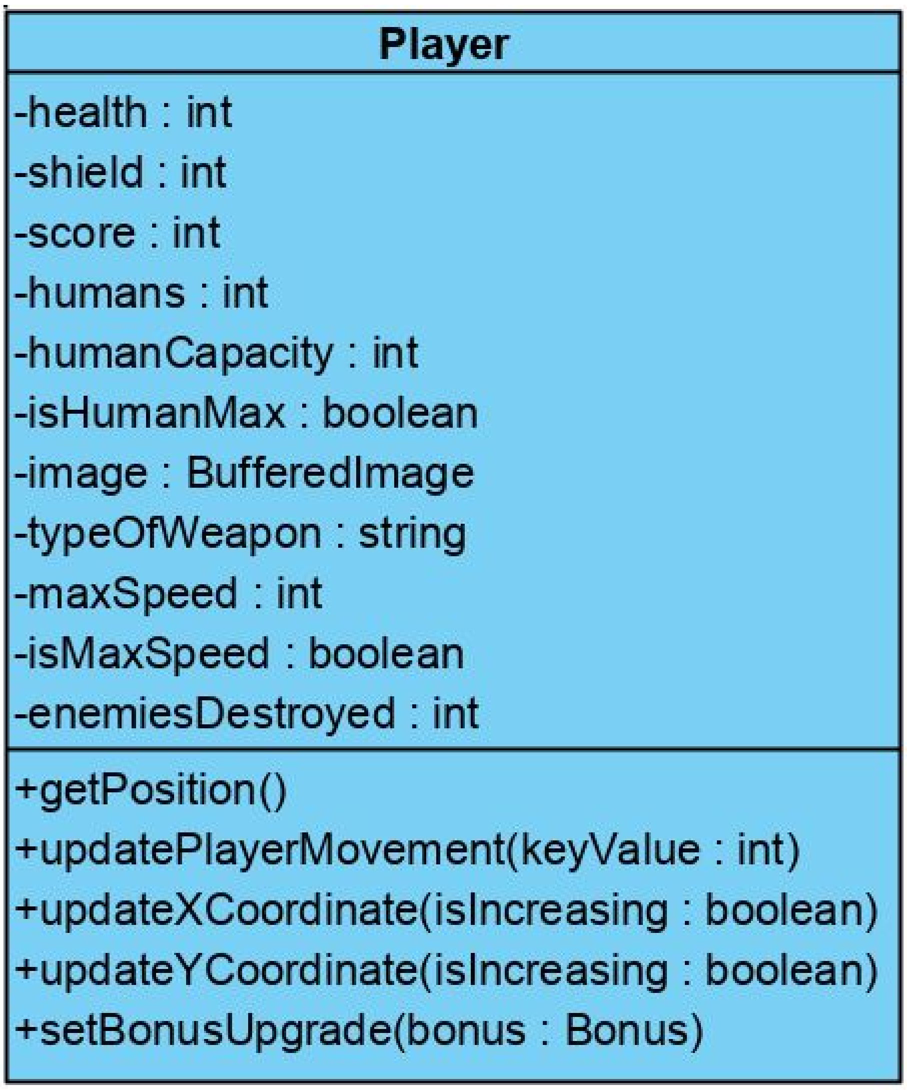
Piece 2 : Enemy Design



Piece 3: Bonus Design



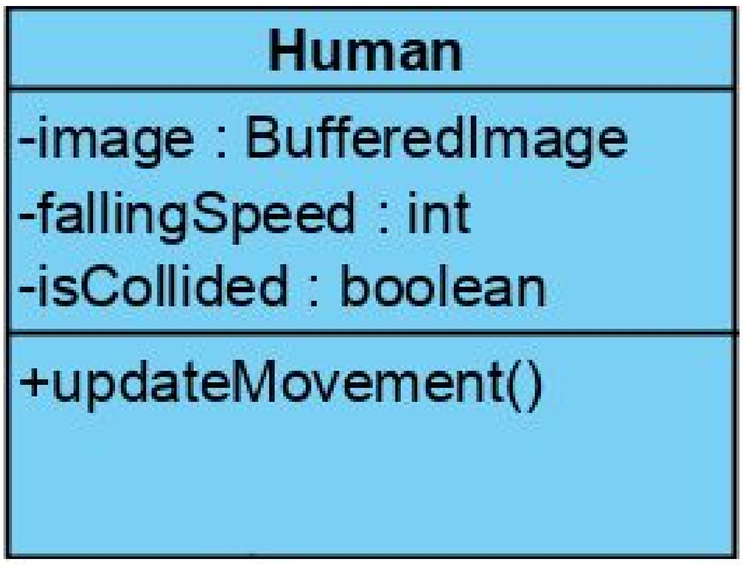
Piece 4: Player Design



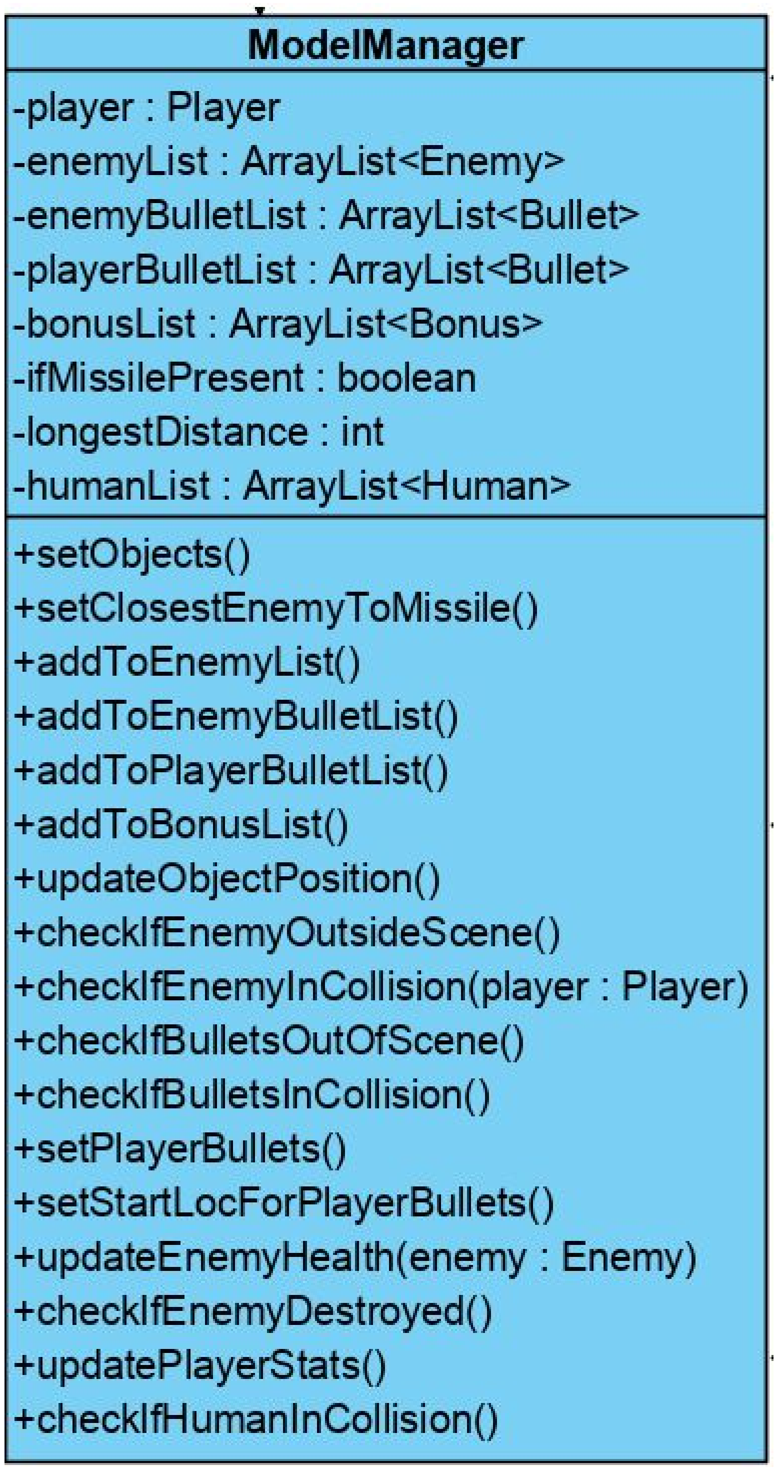
-health : int
-shield : int
-score : int
-humans : int
-humanCapacity : int
-isHumanMax : boolean
-image : BufferedImage
-typeOfWeapon : string
-maxSpeed : int
-isMaxSpeed : boolean
-enemiesDestroyed : int

+getPosition()
+updatePlayerMovement(keyValue : int)
+updateXCoordinate(isIncreasing : boolean)
+updateYCoordinate(isIncreasing : boolean)
+setBonusUpgrade(bonus : Bonus)

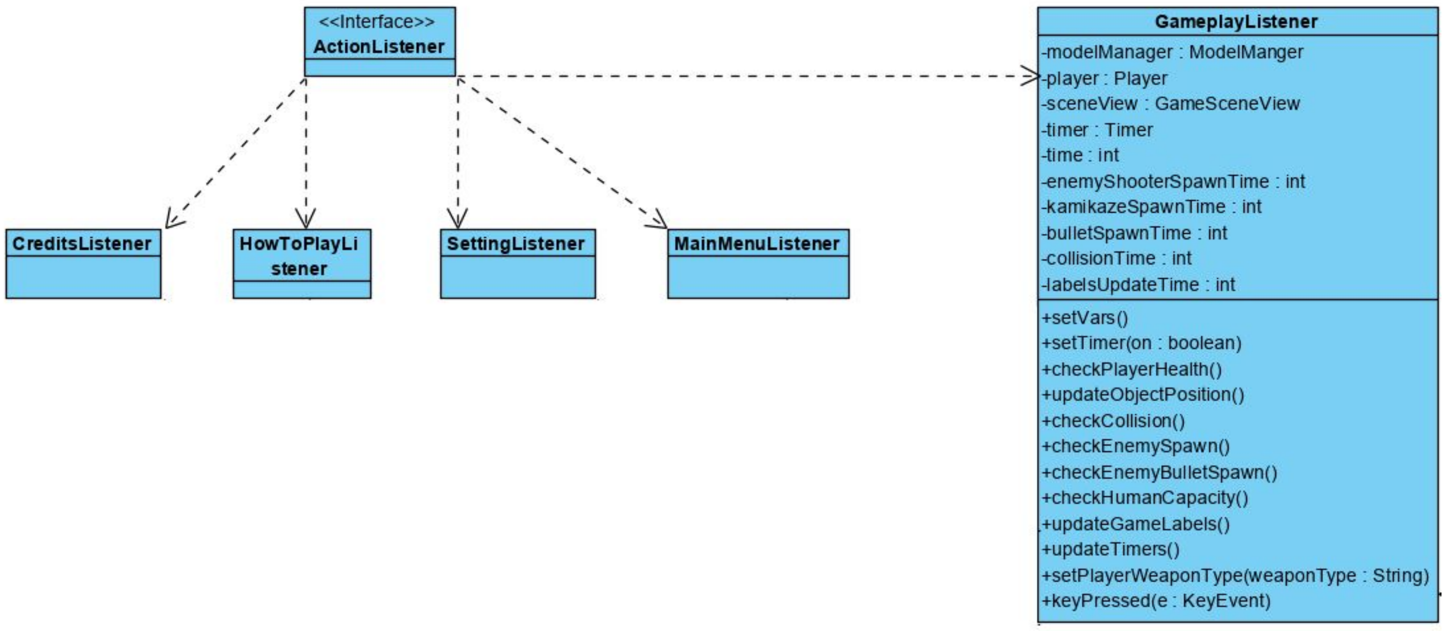
Piece 5: Human Design



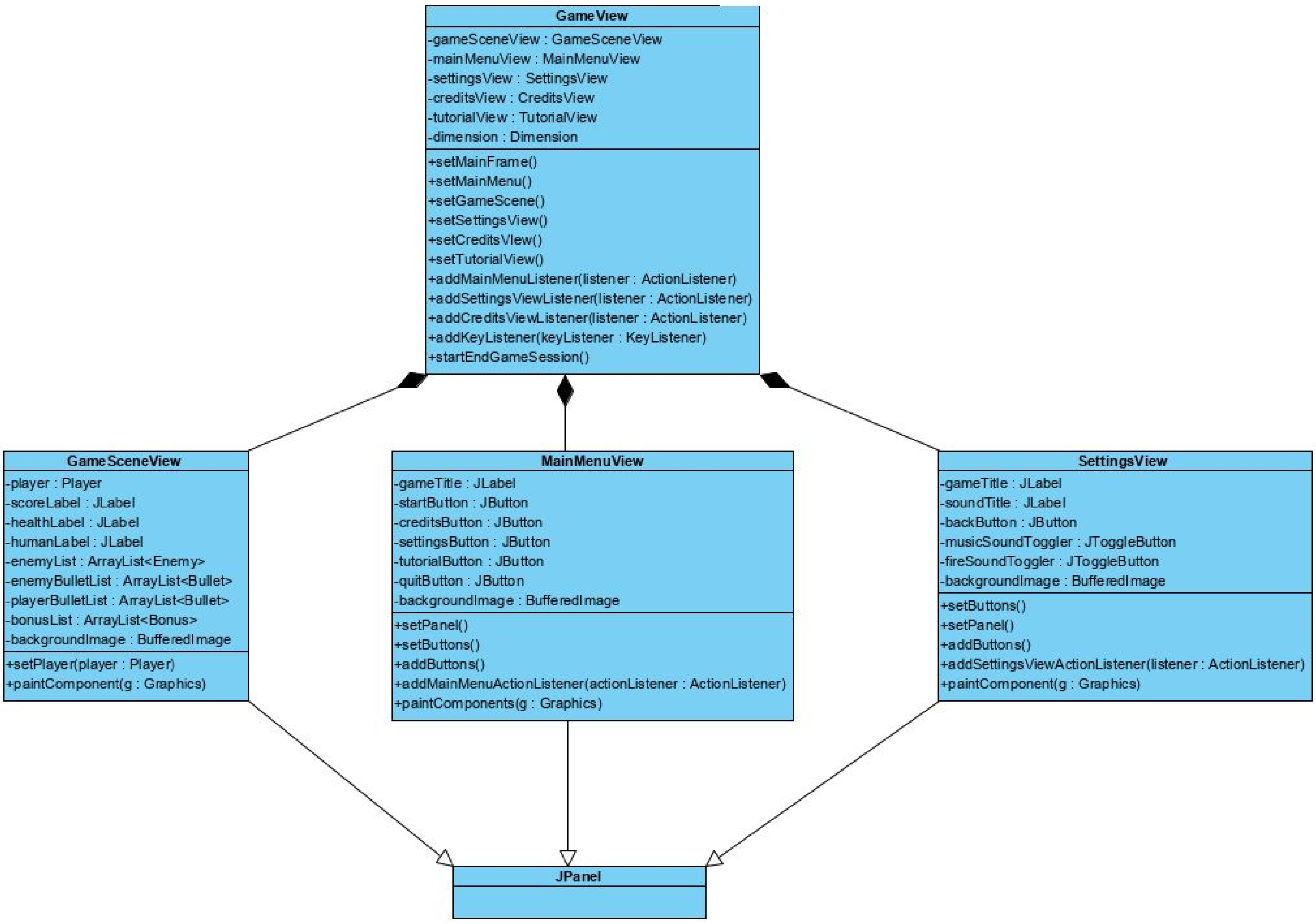
Piece 6: Model Manager Design


ModelManager
-player : Player
-enemyList : ArrayList<Enemy>
-enemyBulletList : ArrayList<Bullet>
-playerBulletList : ArrayList<Bullet>
-bonusList : ArrayList<Bonus>
-ifMissilePresent : boolean
-longestDistance : int
-humanList : ArrayList<Human>
+setObjects()
+setClosestEnemyToMissile()
+addToEnemyList()
+addToEnemyBulletList()
+addToPlayerBulletList()
+addToBonusList()
+updateObjectPosition()
+checkIfEnemyOutsideScene()
+checkIfEnemyInCollision(player : Player)
+checkIfBulletsOutOfScene()
+checkIfBulletsInCollision()
+setPlayerBullets()
+setStartLocForPlayerBullets()
+updateEnemyHealth(enemy : Enemy)
+checkIfEnemyDestroyed()
+updatePlayerStats()
+checkIfHumanInCollision()

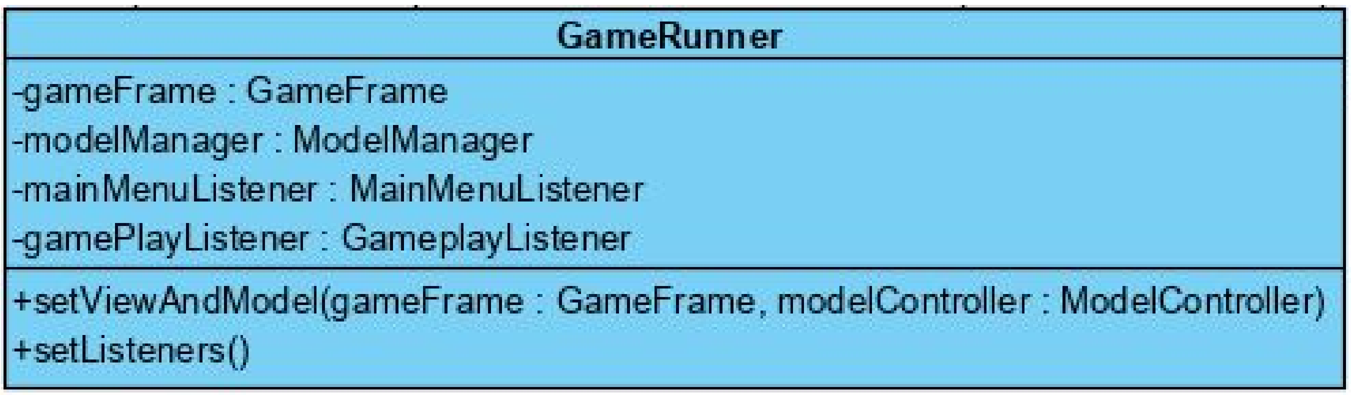
Piece 7: Action Listener Design



Piece 8: Game View Design


GameView
-gameSceneView : GameSceneView
-mainMenuView : MainMenuView
-settingsView : SettingsView
-creditsView : CreditsView
-tutorialView : TutorialView
-dimension : Dimension
+setMainFrame()
+setMainMenu()
+setGameScene()
+setSettingsView()
+setCreditsView()
+setTutorialView()
+addMainMenuListener(listener : ActionListener)
+addSettingsViewListener(listener : ActionListener)
+addCreditsViewListener(listener : ActionListener)
+addKeyListener(keyListener : KeyListener)
+startEndGameSession()
GameSceneView
-player : Player
-scoreLabel : JLabel
-healthLabel : JLabel
-humanLabel : JLabel
-enemyList : ArrayList<Enemy>
-enemyBulletList : ArrayList<Bullet>
-playerBulletList : ArrayList<Bullet>
-bonusList : ArrayList<Bonus>
-backgroundImage : BufferedImage
+setPlayer(player : Player)
+paintComponent(g : Graphics)
MainMenuView
-gameTitle : JLabel
-startButton : JButton
-creditsButton : JButton
-settingsButton : JButton
-tutorialButton : JButton
-quitButton : JButton
-backgroundImage : BufferedImage
+setPanel()
+setButtons()
+addButtons()
+addMainMenuActionListener(actionListener : ActionListener)
+paintComponents(g : Graphics)
SettingsView
-gameTitle : JLabel
-soundTitle : JLabel
-backButton : JButton
-musicSoundToggler : JToggleButton
-fireSoundToggler : JToggleButton
-backgroundImage : BufferedImage
+setButtons()
+setPanel()
+addButtons()
+addSettingsViewActionListener(listener : ActionListener)
+paintComponent(g : Graphics)
JPanel

Piece 9: Game Runner Design


GameRunner
-gameFrame : GameFrame
-modelManager : ModelManager
-mainMenuListener : MainMenuListener
-gamePlayListener : GameplayListener
+setViewAndModel(gameFrame : GameFrame, modelController : ModelController)
+setListeners()

4.2. Design Decisions-Design Patterns

When developing Galaxy Invaders we decided to use MVC (Model-view-controller). Because Galaxy Invaders is an interactive game and depends on both model and controller heavily, MVC will become more useful. One of the main benefits is that MVC can be handled easily within group because it’s divided mainly into 3 parts. Another benefit is that because all the models are grouped together, it will be easy to modify and extend the code. Modifying the code in one component will not create the need to change the code in other components. However, one of the most significant drawbacks is that it can add complexity to code and the rules of the design are not flexible.

4.2.1. Facade Design Pattern

Facade design pattern is used in GameView layer. GameView provides services for GameRunner layer. Also GamePlayLister class serves as facade class. It provides services for GameRunner too.

4.2.2. Composite Design Pattern

In Bullet, Bonus and Enemy classes, composite design pattern is used. ModelManager class stores the list of bullet, enemy, and bonus.

4.3. Packages

We are using two kinds of packages in our implementation. The first ones have been defined by us and the second one is introduced by external libraries such as Java Swing.

4.3.1. Packages From Developers

4.3.1.1. Model Package:

Contains the model classes which include the objects in the game such as spaceships.

4.3.1.2. Controller Package:

Contains the Controller classes which is responsible for controlling the Models and the view panels by defining how the ActionListeners handle those objects.

4.3.1.3. View Package:

Contains the classes which define the User Interface of the app such as Main Menu screen.

4.3.2. Packages From External Libraries

- java.awt.Component
- java.awt.event.ActionEvent
- java.awt.event.ActionListener
- java.awt.event.*
- java.util.ArrayList
- javax.swing.Timer
- java.awt.image.BufferedImage
- java.util.HashMap
- java.awt.Dimension
- java.awt.Point

- java.io.IOException
- javax.imageio.ImageIO
- javax.swing.JFrame
- java.awt.event.KeyListener
- javax.swing.JPanel

4.4. Class Interfaces

4.4.1. GameRunner

attributes:

- **gameFrame:** The main frame of the game.
- **modelManager:** Model class which controls the Model objects of the game.
- **mainMenuListener:** listener which clicks events from MainMenu.
- **gameplayListener:** listener which controls events from the gameplay.
- **settingsMenuListener:** listener which clicks events from Settings.
- **creditsMenuListener:** listener which clicks events from the Credits.
- **helpMenuListener:** listener which clicks events from the Help Menu.

operations:

- **setViewAndModel:** initializes the gameFrame and modelManager objects.
- **setListener:** Sets and assigns all the listeners to the View class objects.

4.4.2. GameplayListener

attributes:

- **timer:** Timer class object to produce ticks every millisecond to call invoke event listeners.
- **time:** Keeps track of time passed.
- **modelManager:** instance of model Manager class to control the Model objects.
- **player:** Instance of the player ship.
- **sceneView:** Panel on which the game runs.
- **enemyShooterSpawnTime:** Time passed since the last spawned enemy shooter.
- **kamikazeSpawnTime:** Time passed since the last spawned kamikaze type enemy.
- **bulletSpawnTime:** Time passed since the last spawned enemy bullet.
- **collisionTime:** Time passed since the last collision check.
- **labelsUpdateTime:** Time passed since the last update of display labels.

operations:

- **setVars:** Initializes the attribute.
- **setTimer:** Starts and stops the timer.
- **checkPlayerHealth:** Ends game of player health is zero.
- **updateObjectPosition:** Calls methods to calculate the position of objects and then updates them accordingly.

- **checkCollision:** Determines if the player is colliding with the enemies or the enemy bullets. Also checks if the enemies are colliding with player bullets.
- **checkEnemySpawn:** Calls the spawn enemy method if a certain amount of time has passed after the last enemy spawn.
- **checkEnemtBulletSpawn:** Calls the spawn enemy bullet method if a certain amount of time has passed after the last enemy bullet spawn.
- **checkHumanCapacity:** Checks if the player's human carrying capacity has been reached. Doesn't allow player to add more humans if so.
- **updateGameLabels:** Updates the stats display such as health and score.
- **updateTimers:** Increments the variables which store the amount of passed time.
- **setPlayerWeaponType:** sets the type of weapon for player.
- **keyPressed:** Updates player position or causes the player ship to fire after determining the key that has been pressed.

4.4.3. GameFrame

attributes:

- **gameSceneView:** Game Scene panel.
- **mainMenuView:** Main menu panel object.
- **settingsView:** Settings panel object.
- **creditsView:** Credits panel object.
- **tutorialView:** Tutorial panel object.
- **dimension:** Used to determine the position of the game frame.

operations:

- **setMainFrame:** Initializes the main frame of the application.
- **addMainMenuListener:** Adds the given ActionListener to the main menu panel.
- **addSettingsViewListener:** Adds the given ActionListener to the settings panel.
- **addCreditsViewListener:**Adds the given ActionListener to the credits panel.
- **addKeyListener:** Adds the given key listener to the Main Frame.
- **startEndGameSesssion:** Hides the main menu panel and makes the game scene panel visible when Start Game button has been clicked and vice versa.

4.4.4. GameSceneView

attributes:

- **player:** Represents the player spaceship.
- **scoreLabel:** Label to represent the player score.
- **healthLabel:** Label to represent the amount of player hp remaining.
- **humanLabel:** Label to represent the amount of humans occupying in the player spaceship.
- **enemyList:** List containing enemy objects.
- **enemyBulletList:** List containing enemy bullet objects.
- **playerBulletList:** List containing player bullet objects.
- **bonusList:** List containing the bonuses.

- **backGroundImage:** Background image for the game scene panel.

operations:

- **paintComponent:** Paints all of the objects and the labels on the game scene panel such as the enemies, the bullets and the labels.

4.4.5. MainMenuView

attributes:

- **gameTitle:** Label representing title of game.
- **startButton:** JButton object to start game.
- **creditsButton:** JButton object to open credits panel.
- **settingsButton:** JButton object to open settings panel.
- **tutorialButton:** JButton object to open tutorial panel.
- **quitButton:** JButton object to quit the application.
- **backGroundImage:** Background image for panel.

operations:

- **setPanel:** Initializes the game panel.
- **setButtons:** Initializes the button objects.
- **addButtons:** Adds the buttons to the panel.
- **addMainMenuActionListener:** Adds an action listener to the button objects.
- **paintComponents:** Paints the label and the buttons on the panel.

4.4.6. SettingsView

attributes:

- **gameTitle:** JLabel object which represents the title of game.
- **soundTitle:** JLabel to represent the sound settings section below.
- **backButton:** JButton to take the user back to the main menu.
- **musicSoundToggler:** JToggler to turn the game music on or off.
- **fireSoundToggler:** Jtoggler to toggle the gameplay effects.
- **backgroundImage:** Background image for the settings panel.

operations:

- **setButtons:** Initializes the buttons and their attributes.
- **setPanels:** Initializes the settings panel.
- **addButtons:** Adds the buttons to the panel.
- **addSettingsViewActionListener:** Adds the given ActionListener to the Jogglers and the JButton.
- **paintComponents:** Paints the buttons and the background image to the settings panel.

4.4.7. ModelManager

attributes:

- **player:** Represents the player spaceship.
- **enemyList:** List containing enemy objects.
- **enemyBulletList:** List containing enemy bullet objects.

- **playerBulletList:** List containing player bullet objects.

operations:

- **setObjects:** Setting all object models; bullet, player and enemy.
- **setClosestEnemyToMissile**: Setting closest enemy to the missile as missile goes toward the closest enemy.
- **addToEnemyList**: Adding enemies to the enemyList.
- **addToEnemyBulletList**: Adding enemy bullets to the enemyBulletList.
- **addToPlayerBulletList**: Adding player bullets to the playerBulletList.
- **updateObjectPosition**: Updating X and Y coordinates of the objects.
- **checkIfEnemyOutsideScene**: Checking if an enemy is outside of the panel.
- **checkIfEnemyInCollisionScene**: Checking if an enemy has collided with a bullet or the spaceship.
- **checkIfBulletOutsideScene**: Checking if a bullet is outside of the panel.
- **checkIfBulletInCollisionScene**: Checking if a bullet has collided with an enemy.
- **setPlayerBullets**: Setting player bullets to blaster, laser or missile according to bonus upgrade.
- **setStartLocForPlayerBullets**: Setting location of bullets of the player as a starting point .
- **updateEnemyHealth**: Updating enemy health to understand whether it is hit or not.
- **checkIfEnemyDestroyed**: Checking if enemy is destroyed.
- **updatePlayerStats**: Updating player stats like health.

- **checkIfHumanInCollision**: Checking if humans have fallen to the ground or collided with the player's spaceship.

4.4.8. Player

attributes:

- **health**: the Health of the player.
- **shield**: Shield that comes from bonus upgrade.
- **score**: Score that comes from destroyed enemies and collected humans.
- **humans**: Captured humans.
- **humanCapacity**: Capacity of saved humans.
- **isHumanMax**: To see if the human capacity is full.
- **image**: Asset of the player.
- **typeOfWeapon**: Type of weapon ; blaster, laser or missile.
- **maxSpeed**: Maximum speed of the spaceship. It can be changed by speed bonus.
- **isMaxSpeed**: To put limit to the speed.
- **enemiesDestroyed**: Winning condition which shows that all enemies are destroyed.

operations:

- **getPosition**: Getting X and Y coordinates of the player.
- **updatePlayerMovement**: Changing X and Y coordinates of the player in order to move it.
- **updateXCoordinate**: Updating X coordinates of the player.
- **updateYCoordinate**: Updating Y coordinates of the player.

- **setBonusUpgrade**: Setting bonus upgrade if the spaceship collected a bonus.

4.4.9. Bonus

attributes:

- **image**: Assets of the bonuses.
- **bonusType**: Bonus types; Speed , weapon and shield.

operations:

- **setSpawnPosition**: Setting bonuses' spawn position.
- **setPosition**: Setting the position of the bonus as they fall down.

4.4.10. Enemy

attributes:

- **health**: the Health of the enemy ship.
- **speed**: The speed of the enemy ship.
- **isDestroyed**: To see if it destroyed .
- **image**: Assets of the enemies.
- **isCollided**: To see if it collided with a bullet or the player's spaceship.

operations:

- **getPosition**: Getting the position of the enemy.
- **setSpawnPosition**: Setting the position of the enemy.
- **setSize**: Setting the size of the different enemies like boss and kamikaze.

  .
- **updateMovement**: Update the movement of the enemy according to its position.

4.4.11. Bullet

attributes:

- **damage**: Damage of the bullet.
- **speed**: Speed of the bullet.
- **type**: Type of the bullet ; blaster, laser and missile.
- **image**: Asset of the bullet.
- **isCollided**: To see whether it is collided or not.
- **isDestroyed**: To see whether it is destroyed or not.

operations:

- **updateMovement**: Updating the movement of the bullet according to its position.

4.4.12. Missile

attributes:

- **deltaX**: X coordinate of the missile.
- **deltaY**: Y coordinate of the missile.
- **target**: Missile’s target.

operations:

- **setTarget**: Setting a target that the missile follows.
- **setDeltas**: Setting coordinates of the missile.
- **updateMovement**: Updating the movement of the missile according to its position.
- **updateDeltas**: Updating the position of the missile.

4.4.13. Human

attributes:

- **image**: Asset of the human.
- **fallingSpeed**: The speed of falling human.
- **isCollided**: To see whether it is collided or not.

operations:

- **updateMovement**: Updating the movement of the human according to its position.

References

[1] "System Design Document." *System Design Document Template*, https://www.cs.fsu.edu/~lacher/courses/COP3331/sdd.html.