CS 319 - Object-Oriented Software Engineering

Software Design Report

Galaxy Invader

Group 3J
Umer Shamaan
Osman Can Yıldız
Denizhan Yağcı
Cemre Osan
Tanay Toksoy

Table of Contents

1. Introduction

Galaxy Invaders is a 2D platformer shooting game. The player can move and shoot in order to reach the goals and complete the stages of the game.

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

1.1. Purpose of the system

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 was made according to the following concepts.

1.2.1. Trade-offs

Development Time vs Performance:

We decided to implement the game using Java Swing. This was mainly due to us having some experience with it from the first year CS courses. This means that we did not have to spend extra time researching about various aspects of implementation. This gave us more time to focus on the development. We also did not have to worry about other matters which create limitations such as memory leaks. However, this decision impacted the performance

negatively as it wouldn't have been as efficient and polished as games developed using JavaFx or C++.

Space vs Speed:

To make the game as efficient as possible in terms of speed, we sacrificed more memory. We had to keep some of the same variables in different places in the memory to increase performance.

Functionality vs Usability:

We wanted to keep the game as simple as possible so we focused more on usability than functionality. We decided to avoid implementing complex features. This improved usability.

Understandability vs Functionality

As mentioned before, the game is intended to be easy to understand and played. Due to this, we only implemented sound settings in the settings and only a few keyboard keys are needed to play the game. Only a minimal amount of time is needed to understand all the aspects of this game.

Development Time vs User Experience:

As we choose Java swing to develop the game, the game would not look as polished as it would compared to game developed with JavaFx. This decision will affect User Experience negatively as there aren't any ornate features such as explosion animations. However, this made the development of the game more simpler, and hence more time could be focused on developing the crucial features of the game.

1.2.2. Criteria

1.2.2.1. 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. After the game has been opened game, we provides a menu that includes a guidance which is “How To Play” button . The reason is that the consistency of the user interface and clarity of the content of the “How to Play” part which introduces the controls to the user.

1.2.2.2. Performance

Our main goal is to make the system as fast as possible and reducing the waiting time for the player between levels. Timers between firing bullets and enemy spawn time prevent from decreasing FPS(frame per second). The users don’t have to wait for the game to be initialized first. When users click on the play button, the game opens and it is initialized while playing. As the game is not huge, initializing while playing is not a problem for computer processors. Furthermore, some updates, such as collision detection, are not performed after every tick from the timer. They occur after a certain amount of time has passed.

1.2.2.3. Maintainability

For the future version of Galaxy invaders, new features and reworks can be easily added to the game. To clarify, thanks to OOP and MVC, this product can be repaired and improved as an update. Therefore, It can be said that Galaxy invaders can survive in today’s market due to new attractive features that may be added to the

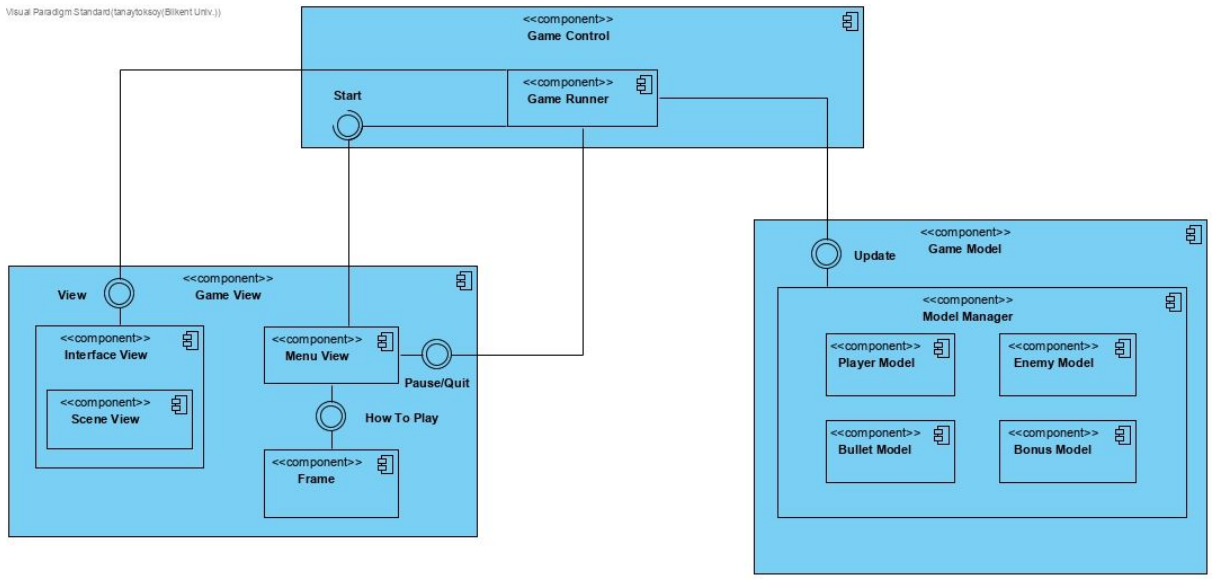
game in the future. Capability enhancement which means enhancing the software to provide new features required by customers and replacing unwanted functionalities to improve adaptiveness and efficiency are our goal for maintainability. Also, it is possible to update some features of the Objects without affecting the other objects. This is thanks to the MVC architectural approach.

1.2.2.4. 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. High-level software 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 ↑, ←, ↓, → 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.

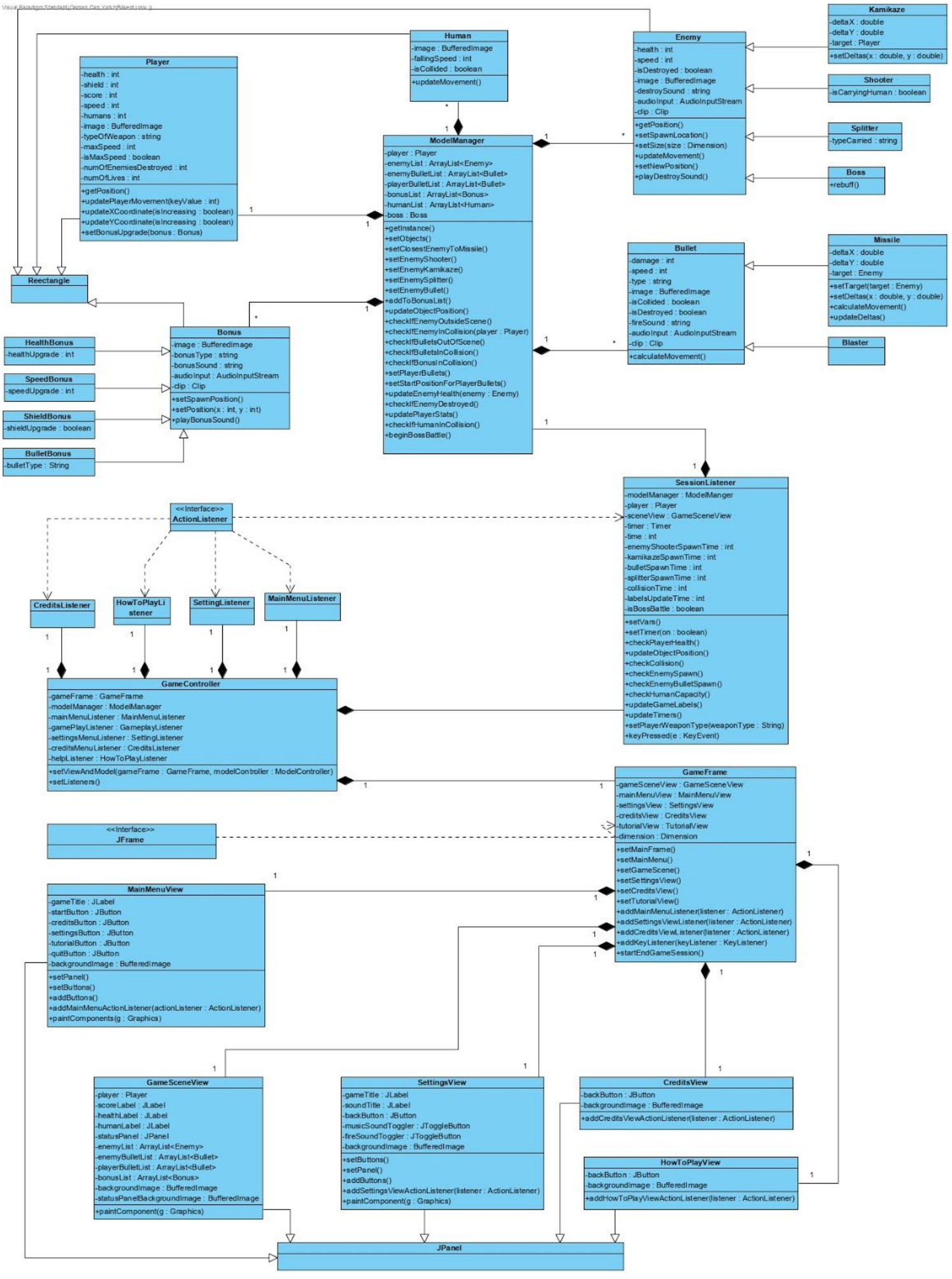
3.3. Game Model Subsystem

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.

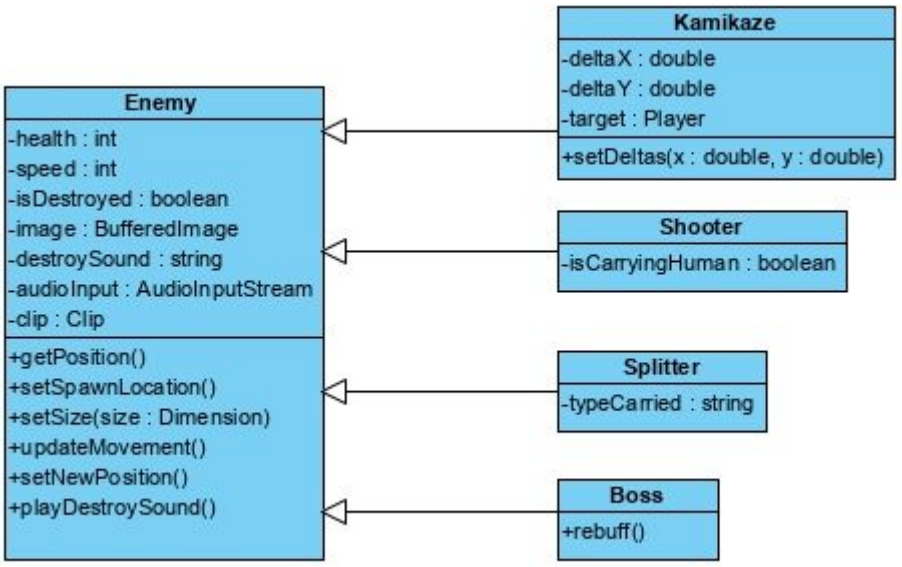
4. Low-level design

4.1. Final object design

Final Object Design is completed as seen below. It includes models, interfaces and classes. To see clearly, we divided the class into pieces.

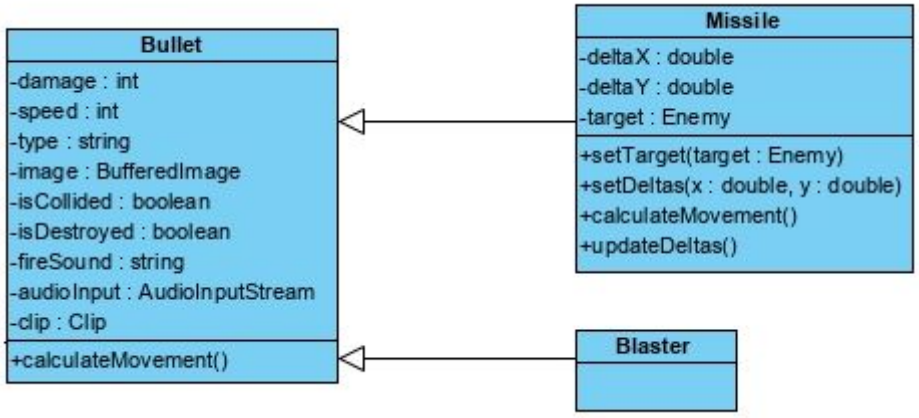

Kamikaze
-deltaX : double
-deltaY : double
-target : Player
+setDeltas(x : double, y : double)
Human
-image : BufferedImage
-fallingSpeed : int
-isCollided : boolean
+updateMovement()
Enemy
-health : int
-speed : int
-isDestroyed : boolean
-image : BufferedImage
-destroySound : string
-audioInput : AudioInputStream
-clip : Clip
+getPosition()
+setSpawnLocation()
+setSize(size : Dimension)
+updateMovement()
+setNewPosition()
+playDestroySound()
Shooter
-isCarryingHuman : boolean
Splitter
-typeCarried : string
Boss
+rebuff()
Player
-health : int
-shield : int
-score : int
-speed : int
-humans : int
-image : BufferedImage
-typeOfWeapon : string
-maxSpeed : int
-isMaxSpeed : boolean
-numOfEnemiesDestroyed : int
-numOfLives : int
+getPosition()
+updatePlayerMovement(keyValue : int)
+updateXCoordinate(isIncreasing : boolean)
+updateYCoordinate(isIncreasing : boolean)
+setBonusUpgrade(bonus : Bonus)
ModelManager
-player : Player
-enemyList : ArrayList<Enemy>
-enemyBulletList : ArrayList<Bullet>
-playerBulletList : ArrayList<Bullet>
-bonusList : ArrayList<Bonus>
-humanList : ArrayList<Human>
-boss : Boss
+getInstance()
+setObjects()
+setClosestEnemyToMissile()
+setEnemyShooter()
+setEnemyKamikaze()
+setEnemySplitter()
+setEnemyBullet()
+addToBonusList()
+updateObjectPosition()
+checkIfEnemyOutsideScene()
+checkIfEnemyInCollision(player : Player)
+checkIfBulletsOutOfScene()
+checkIfBulletsInCollision()
+checkIfBonusInCollision()
+setPlayerBullets()
+setStartPositionForPlayerBullets()
+updateEnemyHealth(enemy : Enemy)
+checkIfEnemyDestroyed()
+updatePlayerStats()
+checkIfHumanInCollision()
+beginBossBattle()
Bullet
-damage : int
-speed : int
-type : string
-image : BufferedImage
-isCollided : boolean
-isDestroyed : boolean
-fireSound : string
-audioInput : AudioInputStream
-clip : Clip
+calculateMovement()
Missile
-deltaX : double
-deltaY : double
-target : Enemy
+setTarget(target : Enemy)
+setDeltas(x : double, y : double)
+calculateMovement()
+updateDeltas()
Blaster
Reectangle
Bonus
-image : BufferedImage
-bonusType : string
-bonusSound : string
-audioInput : AudioInputStream
-clip : Clip
+setSpawnPosition()
+setPosition(x : int, y : int)
+playBonusSound()
HealthBonus
-healthUpgrade : int
SpeedBonus
-speedUpgrade : int
ShieldBonus
-shieldUpgrade : boolean
BulletBonus
-bulletType : String
SessionListener
-modelManager : ModelManger
-player : Player
-sceneView : GameSceneView
-timer : Timer
-time : int
-enemyShooterSpawnTime : int
-kamikazeSpawnTime : int
-bulletSpawnTime : int
-splitterSpawnTime : int
-collisionTime : int
-labelsUpdateTime : int
-isBossBattle : boolean
+setVars()
+setTimer(on : boolean)
+checkPlayerHealth()
+updateObjectPosition()
+checkCollision()
+checkEnemySpawn()
+checkEnemyBulletSpawn()
+checkHumanCapacity()
+updateGameLabels()
+updateTimers()
+setPlayerWeaponType(weaponType : String)
+keyPressed(e : KeyEvent)
<<Interface>>
ActionListener
CreditsListener
HowToPlayListener
SettingListener
MainMenuListener
GameController
-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()
<<Interface>>
JFrame
MainMenuView
-gameTitle : JLabel
-startButton : JButton
-creditsButton : JButton
-settingsButton : JButton
-tutorialButton : JButton
-quitButton : JButton
-backgroundImage : BufferedImage
+setPanel()
+setButtons()
+addButtons()
+addMainMenuActionListener(actionListener : ActionListener)
+paintComponents(g : Graphics)
GameSceneView
-player : Player
-scoreLabel : JLabel
-healthLabel : JLabel
-humanLabel : JLabel
-statusPanel : JPanel
-enemyList : ArrayList<Enemy>
-enemyBulletList : ArrayList<Bullet>
-playerBulletList : ArrayList<Bullet>
-bonusList : ArrayList<Bonus>
-backgroundImage : BufferedImage
-statusPanelBackgroundImage : BufferedImage
+paintComponent(g : Graphics)
SettingsView
-gameTitle : JLabel
-soundTitle : JLabel
-backButton : JButton
-musicSoundToggler : JToggleButton
-fireSoundToggler : JToggleButton
-backgroundImage : BufferedImage
+setButtons()
+setPanel()
+addButtons()
+addSettingsViewActionListener(listener : ActionListener)
+paintComponent(g : Graphics)
CreditsView
-backButton : JButton
-backgroundImage : BufferedImage
+addCreditsViewActionListener(listener : ActionListener)
HowToPlayView
-backButton : JButton
-backgroundImage : BufferedImage
+addHowToPlayViewActionListener(listener : ActionListener)
JPanel

Piece 1: Enemy



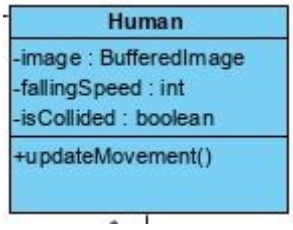
Shooter, Kamikaze, Splitter and Boss extends Enemy class.

Piece 2: Bullet

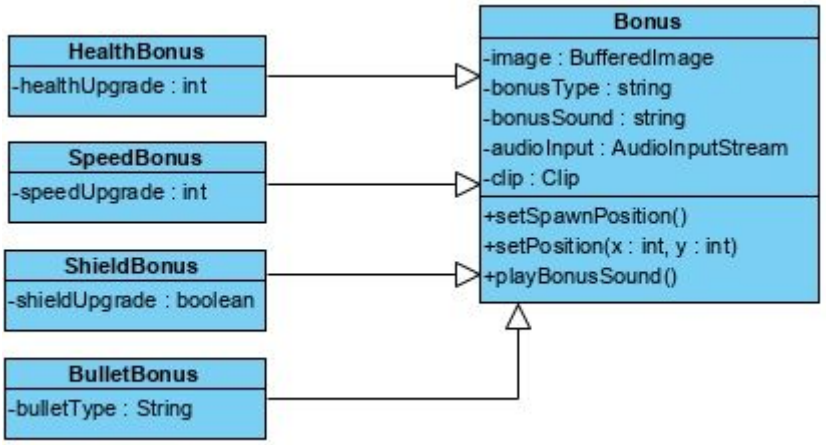


Missile and Blaster extends Bullet class. Blaster Class includes basic bullet and laser bullet which are decided by ModelManager.

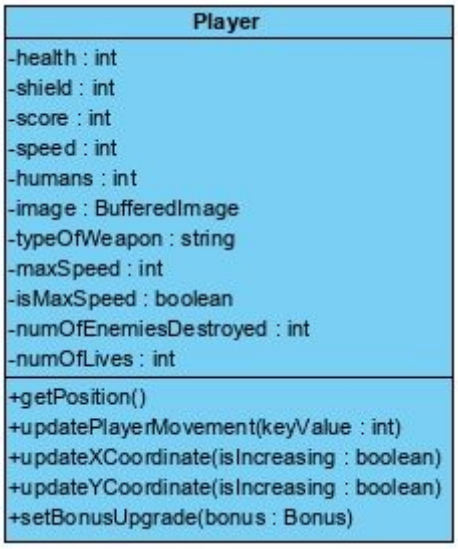
Piece 3: Human



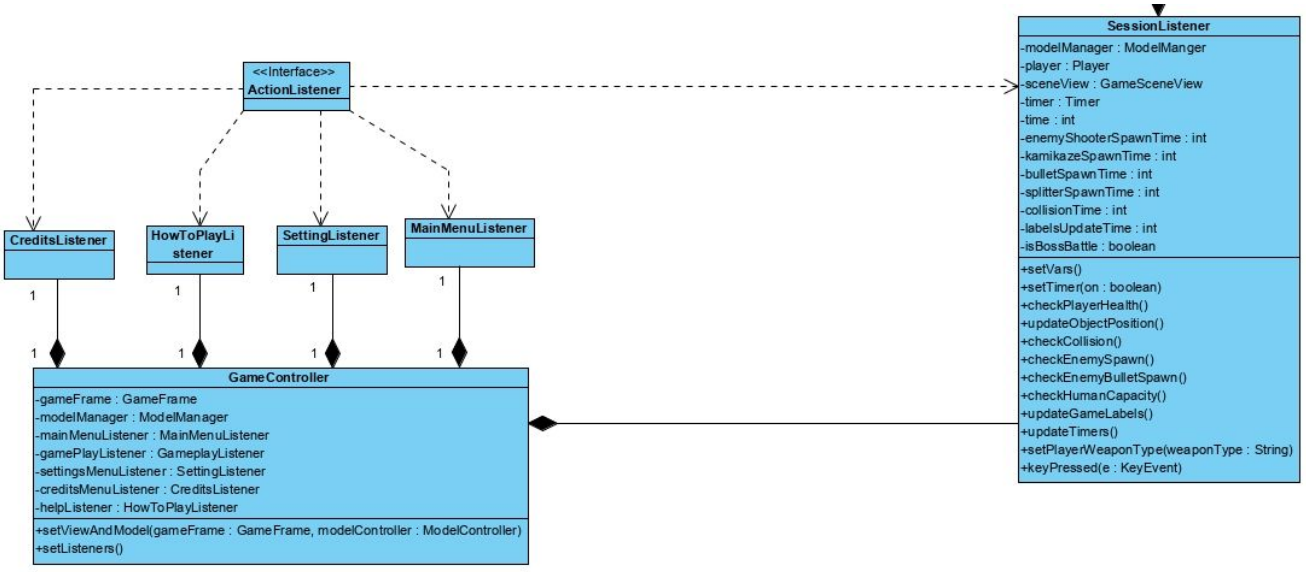
Piece 4: Bonus



Piece 5: Player

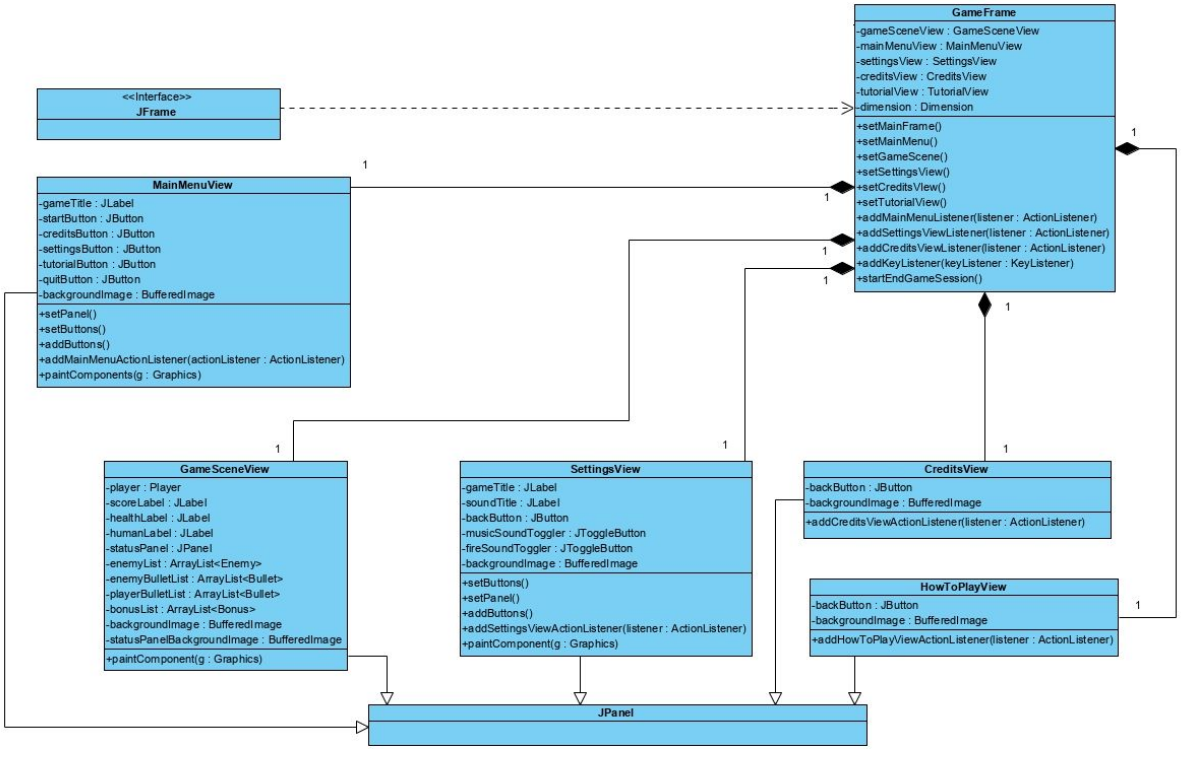


Piece 6: Controller



Piece 7: ModelManager

Piece 8: View



4.2. Design Decision-Design Patterns

We have used quite a few design patterns during the low level implementation. Each of those patterns have their advantages and tradeoffs.

4.2.1. Facade Design Pattern

We have used the Facade Design pattern to reduce coupling between the subsystems. Due to this we have hidden some of the complexities of some subsystems from the client. For example, the Session Listener class can only interact with the Model objects through the model manager. Model manager provides an interface for the Facade Design Pattern.

4.2.2. Singleton Design Pattern

We have used Singleton Design pattern because there will only be one instance of the Model Manager class. This approach will prevent other classes from instantiating their own copies of the Model Manager class. This provides the class with the ability to change the instantiation process.

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 game 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

GameController:

attributes:

- ·**gameFrame:** The main frame of the game holding the panels.
- ·**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.

Session Listener:

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.
- **enemyShooterSpawnTime:** Time passed since the last spawned enemy shooter.
- **kamikazeSpawnTime:** Time passed since the last spawned kamikaze type enemy.
- **splitterSpawnTime:** Time passed since the last spawned splitter 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.
- **isBossBattle:** Checks if there is a boss battle or not. Only operations concerning boss movement and fire are called when true.

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.
- **checkEnemyBulletSpawn:** Calls the spawn enemy bullet method if a certain amount of time has passed after the last enemy bullet spawn.
- **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.

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.
- **setMainMenu:** Initializes the main menu of the application.
- **setGameScene:** Initializes the game scene of the application.
- **setSettingsView:** Initializes the settings of the application.
- **setCreditsView:** Initializes the credits of the application.
- **setTutorialView:** Initializes the tutorial 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.

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.

- **statusPanel:** Contains the labels to present the player with the status of game such as player health.
- **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.
- **statusPanelBackgroundImage:** Background image for status 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.

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.

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.

ModelManager:

attributes:

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

- **bonusList:** List containing bonuses**.**
- **humanList:** List containing human objects.
- **boss:** Represents the bosses spaceship.
- **bossBattle:** Checks if it is a boss battle.

operations:

- **setObjects:** Setting all object models; bullet, player and enemy.
- **getInstance:** Returns the instance of the Class. Prevents the need to instantiate separately.
- **setEnemyShooter:** Setting all EnemyShooters.
- **setEnemyKamikaze:** Setting all EnemyKamikazes.
- **setEnemySplitter:** Setting all EnemySplitters
- **setEnemyBullet:** Setting enemy bullets to blasters.
- **addToBonusList:** Adding Bonuses to bonusList.
- **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.
- **checkIfBulletOutOfScene**: 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.
- **setStartPositionForPlayerBullets**: Setting location of bullets of the player as a starting point .
- **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.
- **beginBossBattle**: Removes the enemies from the scene and spawns the boss.

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.
- **speed**: Movement speed of player.
- **humans**: Captured humans.
- **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.

- **numOfEnemiesDestroyed:** Keeps count of number of enemies destroyed.
- **numOfLives:** The number of number of lives of player. Decrements when player health reaches zero. The game ends when it is decreased to zero.

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.

Bonus:

attributes:

- **image**: Assets of the bonuses.
- **bonusType**: Bonus types; Speed , weapon and shield.
- **bonusSound**: Sound effect for when the bonus picked up.
- **audioInput**: Audio Input Stream.

operations:

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

HealthBonus:

attributes:

- **healthUpgrade:** The amount of health will be added after the health bonus picked up.

SpeedBonus:

attributes:

- **speedUpgrade:** The amount of speed will be added to players current speed after the speed bonus picked up.·

ShieldBonus:

attributes:

- **shieldUpgrade:** The condition for making the players ship shielded or not.

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.
- **destroySound:** Sound will played after enemy spaceship destroyed.
- **audioInput:** The audio input stream that will played for destroySound.

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.

- **setNewPosition**: Setting location of the enemies new position.
- **playDestroySound**: Plays the destroySound when the enemy spaceship destroyed.

Kamikaze:

attributes:

- **deltaX**: The remaining distance for x axis with the target and the enemy kamikaze.
- **deltaY**: The remaining distance for y axis with the target and the enemy kamikaze.
- **target**: The player object that will be chosen to attack.

Operations:

- **setDeltas**: updates x and y coordinates according to player position to update movement.

Shooter:

attributes:

- **isCarryingHuman:** Determines if the shooter will drop human after it is destroyed.

Splitter:

attributes:

- **typeCarried:** Type of enemy carried within.

Boss:

Operations:

- **reBuff**: Shields the boss for a little time.

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.
- **fireSound**: Sound will played after bullet fired.
- **audioInput**: The audio input that will played for fireSound.

operations:

- **calculateMovement:** Calculates the next position of the bullet according to its position.

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.

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.

5. Improvement summary

We mostly have mostly changed the wording in our report according to the comments made by the TA in the last iteration. Furthermore we have also changed a few attributes and a few operations and added some more of them to compensate for some changes or complexities that had arisen in our implementation.

6. Glossary & References

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