CS 319

Object-Oriented Software Engineering

Project Analysis Report

Defender

GROUP 2-D

Taha Khurram - 21701824

Mian Usman Naeem Kakakhel - 21701015

Muhammad Saboor - 21701034

Sayed Abdullah Qutb - 21701024

Balaj Saleem - 21701041

1. Introduction **4**

2. Overview **4**

2.1 Gameplay 4

2.2 Characters 5

2.2.1 Player Spaceship(Al Zarrar) 5

2.2.2 Human 5

2.2.3 Enemy 5

2.2.4 Boss 6

2.3 Pickups 6

2.3.1 Health 6

2.3.2 Fuel 6

2.4 Map and Zones 6

2.4.1 The main map 6

2.4.2 Warp Zone 6

2.5 Events 6

2.6 Game Frame 7

2.7 Rules 7

3 Requirements **7**

3.1 Functional Requirements 7

3.2 Non-Functional Requirements 8

3.2.1 Performance 8

3.2.2 User Interface 8

3.2.3 Player Engagement 8

3.2.4 Extendibility 8

4. System Models **8**

4.1 Use Case model 8

Use Case Descriptions 9

Start the game 9

Control Spaceship with arrow keys 10

Fire Missiles 10

Fire Bullets 11

Clear Screen 12

Get pickup items 12

Open shop 13

Warp and move to another position 13

Quit the game 14

Pause the game 14

Buy Upgrades 15

Resume 15

4.2 Dynamic models 16

4.2.1 Sequence Diagrams 16

4.2.1.1 Sequence Diagram while in Main Menu 16
1 Play 17
2 Shop 17
3 Quit 18
4.2.1.2 Sequence Diagram while in gameplay 18
1 [Esc Pressed] 18
2 [Move key(s) Pressed] 19
3 [Clear Bomb Screen Pressed] 19
4 [Missile Pressed] 19
5 [Bullet Pressed] 19
4.2.2 Activity Diagrams 20
4.2.2.1 Asteroid Generator 20
4.2.2.2 Enemy Generator 20
4.2.2.3 Pickup Generator 21
4.2.2.4. WarpZone Generator 21
Figure 7 22
4.2.2.4. Enemy Mutator 22
4.2.2.5. Fuel Update 23
4.2.2.6. Score Update 23
4.2.2.7 Spaceship Collision 24
4.2.3 State Diagrams 25
4.2.3.1 TitleScreen state diagram 25
4.2.3.2 Coin state diagram 26
4.3 Object and Class model 26
4.4 User Interface - Navigation Paths and Screen Mockups 28

1. Introduction

In the Defender, the Earth is being invaded by aliens and you, as the player, will commandeer a spaceship in an attempt to shoot down all enemy aliens. The aim of the player is to avoid taking damage from the enemy aliens whilst trying to stop them from abducting humans on the surface of Earth. The aliens are abducting humans so that they can absorb their spiritual energy and mutate resulting in a massive boost in power. The aliens aim to capture and kill all humans on the planet as the total spiritual power absorbed will be enough for their entire species to mutate and unlock their full potential becoming extremely powerful with the intention to now destroy the Earth as their are no more surviving humans. It is solely upto the player to save the world and his species from total annihilation.

Defender is an endless runner game which allows the player to control a spaceship and shoot down invading aliens. The player will have to conserve their ammo and prioritize saving the captured humans in order to prevent the aliens from becoming more powerful and consequently destroying the player's home world.

2. Overview

Defender is an endless runner shooter game with a modern Graphical User Interface following the modern norms, the game is designed for both Windows and Linux machines. After starting the game the user will be shown a title screen from where they can decide to start the game or to quit the application and return to the desktop. The player will be able to pause anytime ingame where they will be shown a menu allowing the user to access the shop or resume, and quit the game.

2.1 Gameplay

The game starts with the player already inside the spaceship, the player will us the directional keys to control the spaceship in order to avoid incoming enemy fire while simultaneously trying to shoot at the enemy using the shoot button. The player can choose between three attacks, the first consists of normal bullets, the second will be a heat seeking missile launched towards the enemies, and the third will be a clear screen bomb which will kill all on screen enemies. Some enemies will be trying to abduct humans on ground(located at the bottom of the screen) and the player will have to stop those enemies by shooting at them after which they will drop their human and consequently the player will have to control their spaceship to catch them. If the enemy is able to take the humans to the top of the screen they will get more powerful making them harder to kill.

The game will calculate the player's score as the time spent surviving, and will keep the highest score achieved by the player on top of the screen. The player may also come into contact with warp zones which will randomly transport the player on the map.
If the enemy is able to capture all humans it will result in all enemies getting a power boost and the game will continue until the player is defeated.

The player may choose to pause in game and will have an option to access the shop or resume/quit the game.

The player will earn coins by killing enemies and will be able to spend these coins on upgrades in the shop menu. These upgrades include upgrading the shield or increasing the total fuel tank capacity.

2.2 Characters

Character in the game include the player, humans on the ground and the enemy spaceships.

2.2.1 Player Spaceship(Al Zarrar)

The player will be flying a spaceship named “Al Zarrar”, the following features will be related to the player:

1. Health: The player will have a health percentage and after their health is depleted they will lose the game.
2. Number of missiles: The player will be able to see how many missiles they have left to shoot at enemies.
3. Score: The current score of the player will be shown and updated with the passage of time, the player will also be able to see their highscore on top of the screen.
4. Bullet: The spaceship will fire blue bullets which will damage the enemies.
5. Rocket: The rocket is named “Ghauri Missile” which will be a heat seeker missile and will shoot the closest enemy, the player will have a limited amount of this weapon.
6. Clear Bomb Screen: This weapon will kill all enemies on screen for the player and will also be available in limited amounts.
7. Number of bombs: This will display the number of bombs available to the player for use.
8. Fuel: This will be the amount of fuel left in the spaceship’s tank and if the spaceship runs out of fuel it will crash resulting in the death of the player.

2.2.2 Human

The humans do not interact with the player directly and they will only change states between being on the ground and being abducted. After killing the abducting alien if the player fails to catch the human they will fall to the ground and be killed.

2.2.3 Enemy

The enemy will shoot at the player while trying to capture humans. If the humans are captured they will become stronger and harder to kill. The following information is related to enemies:

1. Health: The enemies will have a value for health which when depleted will result in the enemy being killed.
2. Bullets: The enemy will fire red bolts towards the player.

The game has several different types of enemies who will have varying functions:

1. BugEye: These enemies are responsible for picking up the humans.
2. Crescent: This enemy results after the mutation of weaker enemies and will only attack the player, this is the strongest type of enemy.
3. Saucer: This is a low level enemy which shoots red bullets at the player.
4. Saucer Blades: These are stronger versions of Saucers in terms of speed.

5. Slicer: This is a low level enemy that only attacks the player.
6. Spikey: These enemies move in groups will try to attack the player by getting close and exploding

2.2.4 Boss

The game will feature boss enemies who will appear after a set amount of time and will be considerably stronger than the standard enemies.

There will be two types of bosses:

1. Scythe: This boss will be extremely durable with a heads on approach and will fire bullets at the player.
2. Fighter: This boss will have more damage and will be agile trying to maneuver around the player while firing at the player.

2.3 Pickups

These pickups will provide the user with essential status recovering items such as health and fuel. These items will drop down the screen randomly.

2.3.1 Health

The player will be able to utilize a health pickup which will restore the space ship's health percentage by 10% and if the player's health is already full it will have no effect.

2.3.2 Fuel

The player will use the fuel pickup to restore the fuel percentage to full.

2.4 Map and Zones

2.4.1 The main map

The game will have one fixed map in which the player will navigate their spaceship, there will be a minimap overhead in which a portion of the map is shown in order to show the player their current location on the overall map.

The map will have humans on the bottom of the screen i.e. on ground, and the enemies with the player will be flying in the rest of the screen.

2.4.2 Warp Zone

There will be random warp zones generated on the map will will transport the player from one location to another.

2.5 Events

We will implement an asteroid shower event in which several asteroids will fall down the map and the player has to avoid them in order to assure his/her safety.

2.6 Game Frame

The gameframe will be used as a hud to show several important information tabs in the game:

1. Health Percentage: The spaceship's health percentage will be shown in the gameframe.
2. Score: The player's current score will be shown in the gameframe and will be updated with the passage of time, the player will be able to see their high score as well in the game frame.
3. Missile Count: The number of missiles available to the player will be shown.
4. Map View: A minimap will be shown, this minimap will feature a small section of the map in which the player is located(will be in the centre of this map).
5. Camera View: The player's section of the map will be shown in full with all enemies, humans, bullets fired by both teams and the spaceship on screen.
6. Coins: The number of coins will be shown.
7. Fuel Percentage: The current fuel percentage will be shown and will depict the status of the player's fuel tank.
8. Pickups: Pickups such as health and fuel will also be shown in the game frame.

2.7 Rules

The game follows an endless runner-esque format, in the game the enemies will keep generating throughout however the map will not change and the player will gain score with the passage of time and earn coins by killing enemies. The game will continue on till the player loses all of his health or fuel, however through pickups the player may be able to restore them. The player will also be able to upgrade his spaceship by spending earned coins.

If all the humans are abducted or killed the enemies will gain a power up making them substantially more powerful.

3 Requirements

3.1 Functional Requirements

1. New Game: Player can start a new game in which the player's health, missile count and bomb count will be set to full. The score will be set to zero and increase as the game progresses. Incase of enemies and humans they will also be randomly spawned on the map, and the enemy will keep spawning as the game continues on.
2. Quit: This option will exit the game. The user may also access this option from the pause menu ingame.
3. Pause Menu: The user may enter this menu ingame after pressing the pause button, it will feature the option to continue the game i.e. unpause, shop, or to quit the game and return to the desktop.
4. Shop Menu: The shop menu is located in the pause menu and the player will be allowed to spend their coins earned here on upgrades to make themselves more powerful.

3.2 Non-Functional Requirements

3.2.1 Performance

To ensure an enjoyable game experience the performance of the game is to be kept optimal, without compromising graphics and mechanics. Rendering will be done according to the camera frame, enabling low resource usage and better overall performance. Ideally, the game would run smoothly even on lower end machines, due to the minimalistic nature of the game.

3.2.2 User Interface

Menus, title screens (possibly splash screens) and other such scenes are to be designed in a way to ensure accessibility and user friendliness. Basic start and pause menus are provided to the user to allow continue gameplay at his/her discretion. Menus are expected to be simple and precise.

3.2.3 Player Engagement

Extra features of warp, clear screen and a shop are added to ensure the player engages with the game for longer periods due to increased variation and strategies.

Inorder to create a form of competition the player will have to manage their health and fuel percentages in order to achieve the highest score possible which will be shown at the top of the screen.

Classes of enemies also work towards the same goal, variation in enemies provides a dynamic element to the game which may be intriguing for the user.

3.2.4 Extendibility

The code base is to be written such that new features to the game are easily includable. These extended features may in the future include, improved mechanics, improved graphics, variety of enemies, spaceship selection, planet selection etc. Hence the base classes are to ensure that that these features are added with minimum amount of rework.

4. System Models

4.1 Use Case model

The use case of the game consists of one main (primary) actor which is the player. There is also a secondary actor, which is not controlled by the user but by the computer itself. This secondary actor is the enemy and impacts the game in different ways.

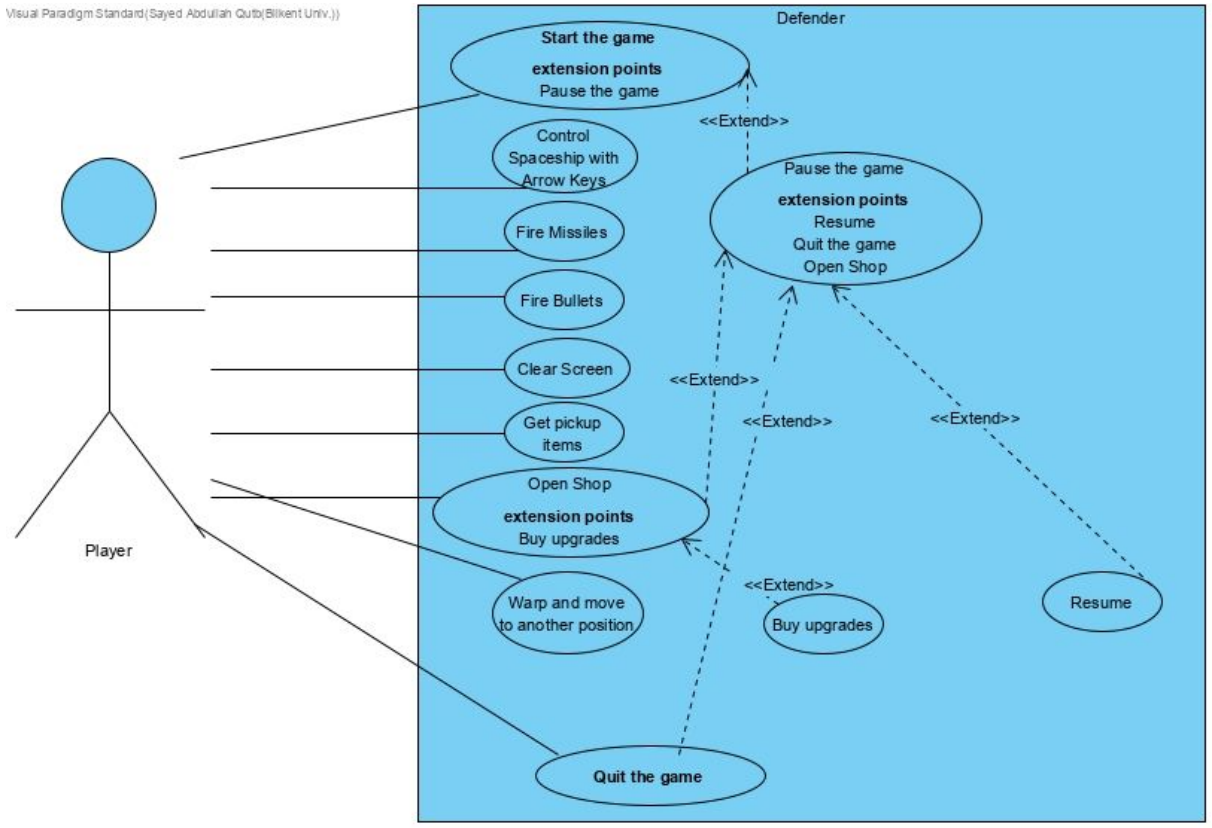


Figure 1 (Use Case Diagram)

Use Case Descriptions

1. Start the game

Name: Start the game

Participating Actor(s): Player

Entry Condition(s):

- Player opens the game

Exit Condition(s):

- Player starts playing and controlling the spaceship in the game

Flow of events:

a. Player opens the game
b. Player clicks on “Start”

c. Game window shows up
d. Player gets to see the spaceship, the enemies, and the humans

Special Requirement(s):

- Player can pause the game by pressing the ESC button on the keyboard.

2. Control Spaceship with arrow keys

Name: Control Spaceship with arrow keys

Participating Actor(s): Player

Entry Condition(s):

- Player starts the game

Exit Condition(s):

- Player ends up controlling the spaceship and might fire missiles
- Spaceship moves from one spot to another

Flow of events:

a. Player opens the game
b. Player clicks on “Start”
c. Game window shows up
d. Player gets to see the spaceship, the enemies, and the humans
e. Player presses any of the arrow keys on the keyboard and the spaceship starts moving

Special Requirement(s): None

3. Fire Missiles

Name: Fire Missiles

Participating Actor(s): Player

Entry Condition(s):

- Player starts the game
- Player presses “Space” key

Exit Condition(s):

- Player fires a missile which travels from the spaceship and hits either an enemy or the missile is wasted
- Missile comes out of the spaceship
- If the missile hits the enemy, player gets coins
- Missile is more powerful than the bullet and causes more damage to the enemy

Flow of events:

a. Player opens the game
b. Player clicks on “Start”
c. Player might want to adjust the spaceship according to the enemies’ position
d. Player presses a key to fire the missile and the missile starts traveling towards the player
e. Missile impacts the enemy if it hits them
f. Player gets coins for successful hits

Special Requirement(s): None

4. Fire Bullets

Name: Fire Bullets

Participating Actor(s): Player

Entry Condition(s):

- Player starts the game
- Player presses “Space”

Exit Condition(s):

- A bullet ends up going from the spaceship towards the enemies’ side
- Bullet either hits the enemy or is wasted
- Successful hits result in coins being awarded to the player

Flow of events:

a. Player opens the game
b. Player clicks on “Start”
c. Player controls the spaceship and moves it according to the enemies’ position
d. Player fires a bullet by pressing the space key on keyboard
e. A bullet is being fired from the spaceship and goes towards the enemies
f. Bullet either hits the enemy or hits nowhere
g. Successful hits gives player coins

Special Requirement(s): None

5. Clear Screen

Name: Clear Screen

Participating Actor(s): Player

Entry Condition(s):

- Player starts the game

Exit Condition(s):

- The game screen ends up being free of enemies
- All enemies are gone for a small amount of time

Flow of events:

a. Player opens the game
b. Player clicks on “Start”
c. Player clicks on “Clear Screen” and the screen is cleared and empty

Special Requirement(s): None

6. Get pickup items

Name: Get pickup items

Participating Actor(s): Player

Entry Condition(s):

- Player starts the game
- Pickup item is dropped from above

Exit Condition(s):

- The player picks up the falling item.
- The spaceship is refueled if the pickup item is fuel.
- The player might also get upgraded missile type.

Flow of events:

a. Player opens the game
b. Player clicks on “Start”
c. Player plays until a pickup item is dropped from the top and the player catches it

Special Requirement(s): Pickup items are dropped at certain time intervals and is not always available

7. Open shop

Name: Open shop

Participating Actor(s): Player

Entry Condition(s):

- Player starts the game
- Player has enough coins to buy an item

Exit Condition(s):

- The player gets to see shop items ready for sale
- The player might buy a shop item if he has enough coins.
- He might buy an upgraded missile or fuel for the spaceship if he has enough coins.

Flow of events:

a. Player opens the game
b. Player clicks on “Start”
c. Player presses “ESC” key and the game menu appears
d. Player clicks on Shop and the shop opens

Special Requirement(s): Player needs coins if he wants to buy items from shop.

8. Warp and move to another position

Name: Warp and move to another position

Participating Actor(s): Player

Entry Condition(s):

- Player starts the game

- Player moves to the warp zone

Exit Condition(s):

- Player eventually comes out of the warp zone in a different position than before.

Flow of events:

a. Player opens the game
b. Player clicks on “Start”
c. Player moves the spaceship towards the warp zone
d. Player enters the warp zone and is transported to a different position in the game screen.

Special Requirement(s): None

9. Quit the game

Name: Quit the game

Participating Actor(s): Player

Entry Condition(s):

- Player opens the game

Exit Condition(s):

- Player exits from the game

Flow of events:

a. Player opens the game
b. Player clicks on “Quit”

Special Requirement(s): None

10. Pause the game

Name: Pause the game

Participating Actor(s): Player

Entry Condition(s):

- Player starts the game

Exit Condition(s):

- The game is paused
- Game menu is shown to the player

Flow of events:

a. Player starts the game
b. Player presses “ESC” key to pause the game

Special Requirement(s): None

11. Buy Upgrades

Name: Buy Upgrades

Participating Actor(s): Player

Entry Condition(s):

- Player starts the game
- Player pauses the game by pressing “ESC” key

Exit Condition(s):

- Player is equipped with upgrades

Flow of events:

a. Player starts the game
b. Player presses “ESC” key to pause the game
c. Player opens Shop
d. Player clicks on a shop item and buys if he has enough coins

Special Requirement(s): Player needs coins to buy upgrades

12. Resume

Name: Resume

Participating Actor(s): Player

Entry Condition(s):

- Player starts the game
- Player pauses the game by pressing “ESC” key

Exit Condition(s):

- Game on!

Flow of events:

a. Player starts the game
b. Player presses “ESC” key to pause the game
c. Player clicks on “Resume”
d. Game is continued again

Special Requirement(s): None

4.2 Dynamic models

4.2.1 Sequence Diagrams

The game’s models consist of 2 sequence diagrams, one is for the main menu and the other one is for when the player is playing the game.

4.2.1.1 Sequence Diagram while in Main Menu

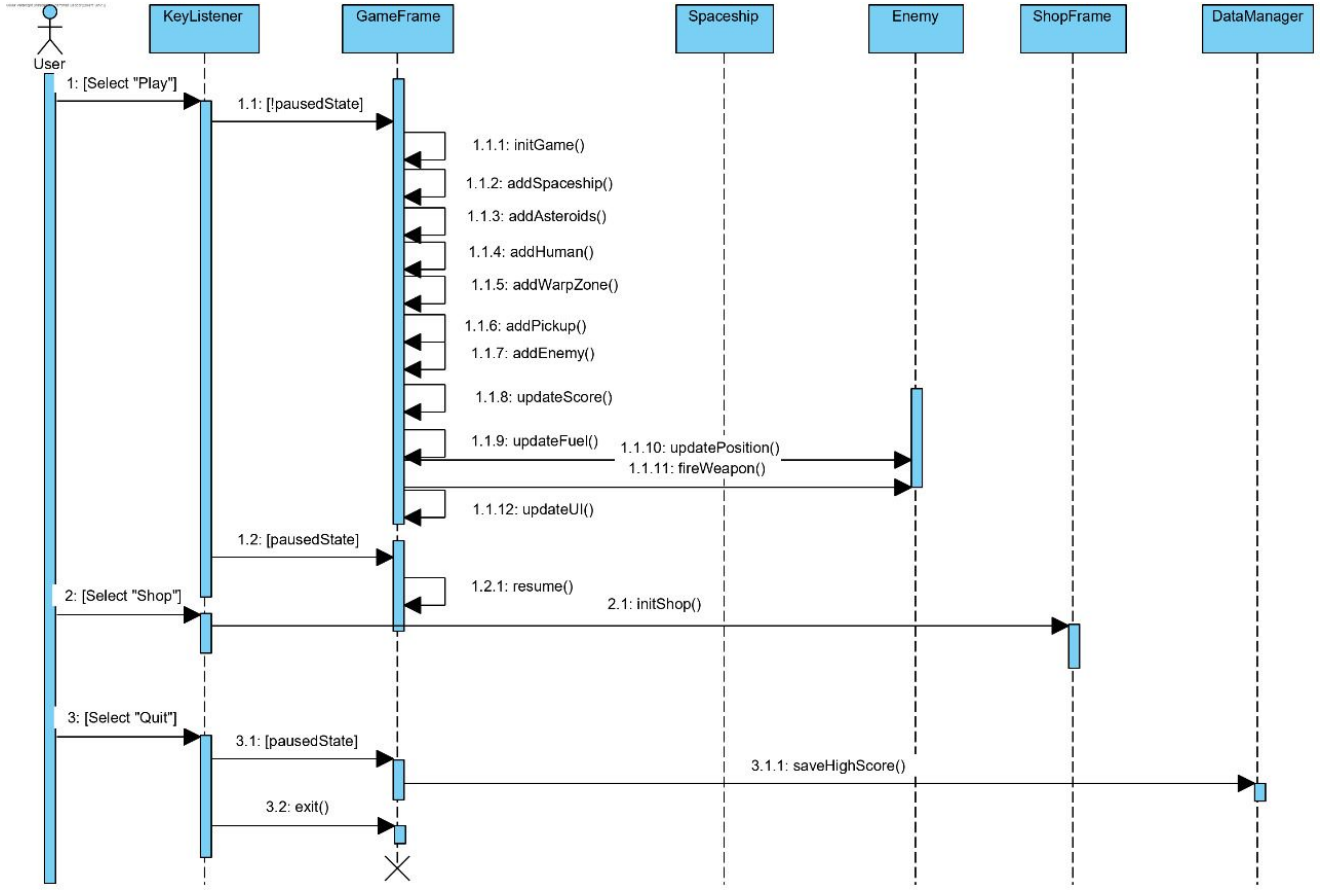


Figure 2 (Sequence Diagram 1)

This Sequence diagram details the list of actions the user can take while present in the main menu.
The following explain in detail all the actions the user can take and how they will be implemented:

1 Play

When the player clicks the play button a keylistener will be used to determine the user's actions and the player will be sent inside the gameframe where the user will start a new game.

1. ![pausedState]:
    a. initGame(): this method will be called upon initialization to start a new game for the user.
    b. addSpaceship(): this method will generate the player's spaceship on the map.
    c. addAstroids(): the addAstroids() may be called randomly in the game at anytime and it will summon a falling asteroid in the game which the player will avoid colliding into.
    d. addHuman(): the game will call this method upon initialization and this will randomly spawn humans on the map.
    e. addWarpZone(): this method will randomly spawn a warp zone on the map which upon collision will warp the user to the end location preset. The game will randomly call this method and may call this method several times throughout the game.
    f. addPickup(): this method will be called throughout the game and will be used to generate the health and fuel pickups for the player. The method will be called more often when the player's fuel or health stats are low.
    g. addEnemy(): the game will first call this method upon initialization to generate enemies for the player to kill after which this method will be called to keep generating enemies for the player.
    h. updateScore(): the aim of the player is to survive the incoming enemies hence this method will be called with time throughout the game to increase the current score of the player.
    i. updateFuel(): this method will be called throughout the game as the player will constantly be using fuel and may use a pickup to refuel.
    j. updatePosition(): this method will be called upon the creation of an enemy and will be used to update the enemy's position throughout the game.
    k. fireWeapon(): when the player will be in the enemy's line of sight this method will be called so that the enemy will shoot their weapons at the player.
    l. updateUI(): whenever any movement occurs ingame, this method will be called in order to update the UI so that the frontend matches with the backend.
2. [pausedState]:
    When the user presses the pause button ingame, they will be sent to the pause menu where they may choose to go back to the gamy by pressing the resume button which will call the resume() method.

2 Shop

If the player selects the shop option it will be detected by using a keylistener and the user will be sent to the shop frame where they will be able to purchase upgrades for their spaceship.

3 Quit

When the user selects the quit option the game will check if the user was in the pausedState or quitting at the beginning of the game and will accordingly call the following methods:

1. [pausedState]: If the user was in the paused state and selected the quit option the game will check the user's current score and if it is higher than their high score the game will store this score as the new highscore in the game by calling the saveHighScore() method and then exiting the game.
2. exit(): If the user was in the main screen when they selected to quit the game the exit() method will be called and the user will exit the game and the gameframe object will be deleted.

4.2.1.2 Sequence Diagram while in gameplay

While the player is playing the game, the player has a bunch of options he can use. He can control and move the spaceship with the Arrow Keys, fire laser missiles at the enemies, clear all of the enemies from the screen, use different types of missiles, such as Mass Destruction Missile which kills all the enemies. Also the player can pause the game with the ESC button and take a break, or quit the game in whole.

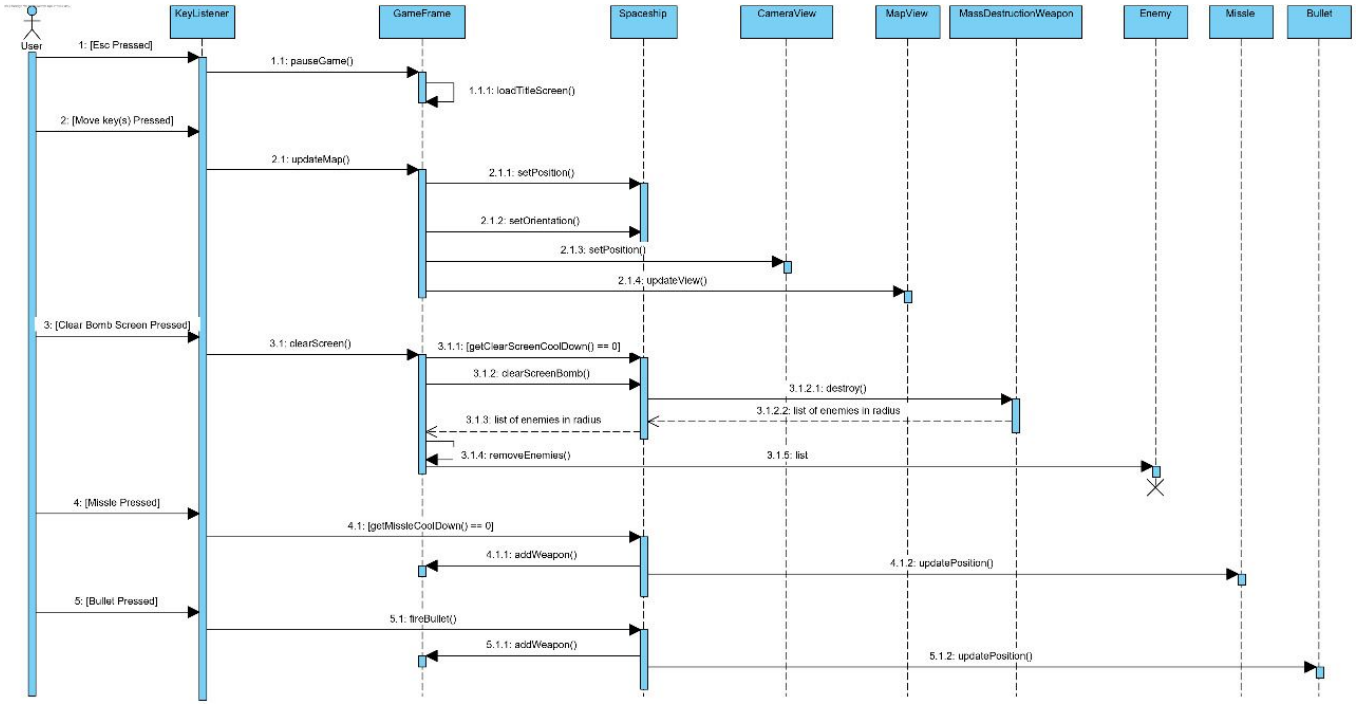


Figure 3 (Sequence Diagram 2)

The following explains sequence diagram 2 in detail:

1 [Esc Pressed]

When the user pressed the esc key the keybinder will detect this and accordingly call the pauseGame() method after which the user will be taken the the title screen by calling the loadTitleScree() method.

2 [Move key(s) Pressed]

When the user presses any of the movement keys the game will call the updateMap() method which will then calculate the new position of the spaceship by calling the setPosition() method and if the user changed directions it will then call the setOrientation() method and calibrate the new orientation of the spaceship.

Incase of the camera viewpoint the movement of the ship will be shown on the camera and the camera will adjust its position accordingly by calling the setPosition().

Furthermore, to show the changes on screen the game will call the updateView() method to update the UI on the map.

3 [Clear Bomb Screen Pressed]

If the player presses the Clear Screen Bomb button the game will call the clearScreen() method which is meant to kill all enemies on screen by using the following methods:

1. [getClearScreenCoolDown() == 0]: After the user presses the clear screen bomb button the game will check if the user has access to this bomb and if the cooldown is zero then the next method will be called otherwise the user will not be able to use this bomb.
2. clearScreenBomb(): when the count == 0 is true, this method will be called which will use the clear screen bomb. Following this method the destroy() method will be called which will get the list of enemies in radius and they all will be killed.
3. removeEnemies(): After the bomb has been deployed and enemies killed this method will be used to clear the list of enemies and delete the enemy object.

4 [Missile Pressed]

1. [getMissileCoolDown() == 0]: When the user presses the missile button, the getMissileCoolDown() method will be called which will check if the user has a missile to use right now.
2. addWeapon(): this method will be called to add the missile fired to the gameframe so that it will be shown on screen.
3. updatePosition(): this method will be used to follow the missiles movement and update them on screen so that we can see the missile moving on screen as well.

5 [Bullet Pressed]

1. fireBullet(): When the user presses the fire bullet button this method will be called.
2. addWeapon(): Once the fireBullet() method has been called this method will be called to add the fired bullet onto the game frame.
3. updatePosition(): With the bullet having a set speed and having a visual presence on the game frame this method will be used to show the bullet moving on screen.

4.2.2 Activity Diagrams

4.2.2.1 Asteroid Generator

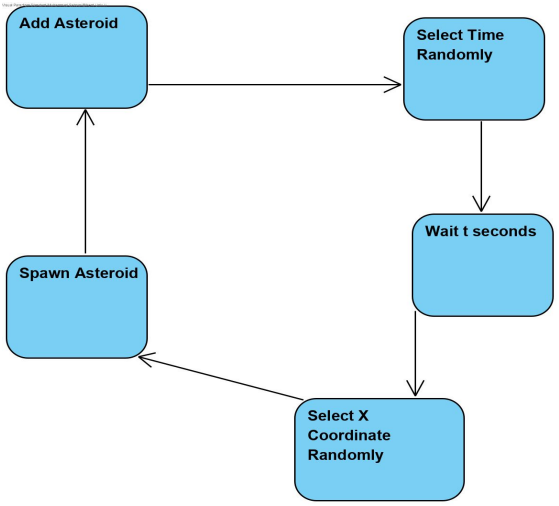


Figure 4

This activity diagram depicts the transition of events to create an asteroid in game. Firstly the add asteroid method will be called which will start this chain after a time will be selected randomly after which the asteroid is to be deployed. Once the wait time has ended the system will select a random x coordinate on top of the map where the asteroid will be spawned and this will continue throughout the game.

4.2.2.2 Enemy Generator

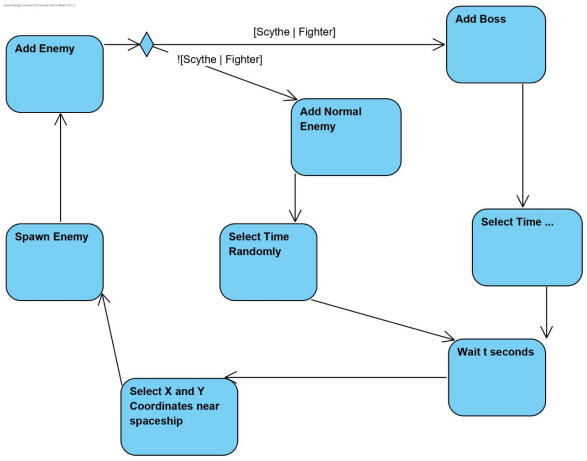


Figure 5

This activity diagram shows the steps in which an enemy or a boss unit is spawned. First the add enemy method is called after which we have two options:

1. The enemy to be spawned is a boss being a Scythe or a Fighter unit which will result in an add boss event. Following this a time will be selected after which the boss is to be spawned.

After the ‘t’ amount of time has passed an X and Y coordinate close to the spaceship will be selected and the enemy will be spawned.

2. If a normal type of enemy is being spawned then the game will not select a boss type instead it will spawn a normal type enemy. First the game will select a time randomly and after that time has passed the game will select the X and Y coordinates close to the user and spawn the enemy.

4.2.2.3 Pickup Generator

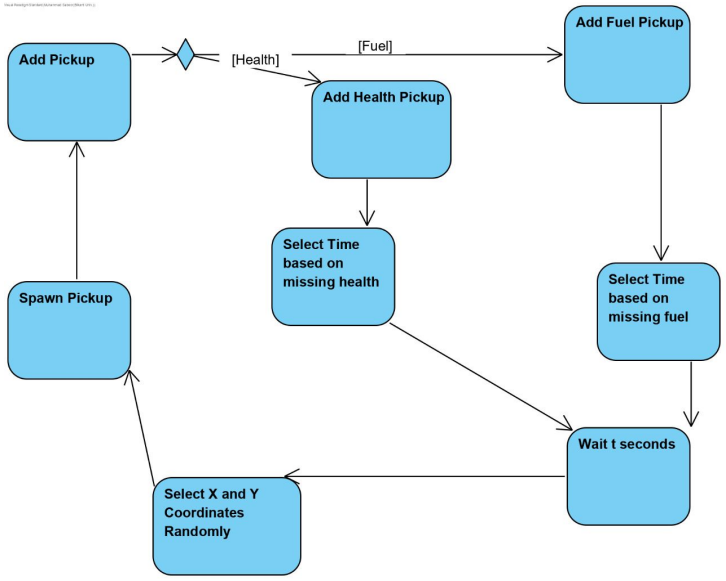


Figure 6

Add pickup is an activity concerning the random generation of pickup items on our map. Simply put these items will be dispensed at random locations and the frequency of these spawns will totally depend on the situation that the player is in. The activity starts when the addPickUp() method is called. Since we have 2 kinds of pickups that will be made available to the player during the game, the diagram is divided into 2 parts. If we are dispensing a Fuel item, we will first see the fuel condition of the spaceship and then will apply an algorithm which will calculate the time which we have to wait before dispensing a fuel item. If the fuel is very low then this time will be very low as well and if the fuel is not at a dangerous level, then the time will be a little more than before. Thus creating a proportional relation with current fuel level. The same algorithm is applied for dispensing a health pickup. After this random x and y coords are selected inside the camera view boundary and the pickup is dispensed.

4.2.2.4. WarpZone Generator

One of the features of the game is providing the player with small warp zone type places with which the spaceship can jump to random places on the map. This new random place can have a very different setting than the place where the spaceship warped from i.e it can have more or less enemies than before. When addWarpZone() is called, the system selects the time to wait before adding this warp zone randomly and then after waiting for that specific amount of time, the system selects the random coords on which the warpzone has to be created near the spaceship and then the warpzone is created.

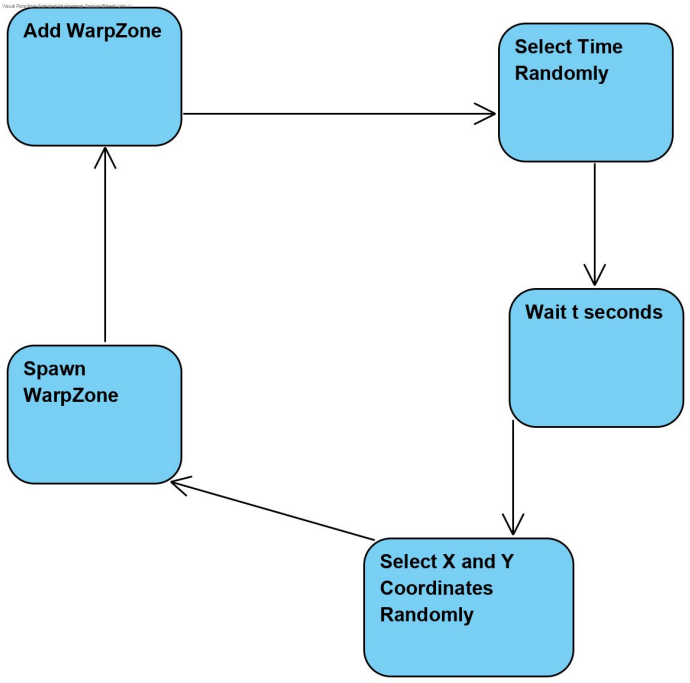


Figure 7

4.2.2.4. Enemy Mutator

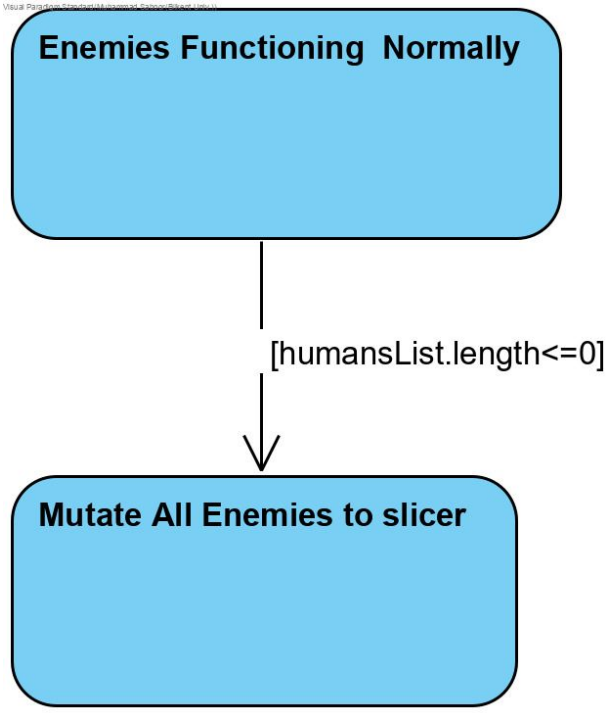


Figure 8

This is a system activity but it has some conditions that must be satisfied before this activity can take place. When this activity is called, it first checks whether all of the humans have been kidnapped by the aliens or have died or not. If even one of the humans remain then this activity will

not place. On the other hand if all of the humans have died, then all of the enemies in the game will get a massive powerup and all of them will mutate into Slicer, the strongest normal enemy.

4.2.2.5. Fuel Update

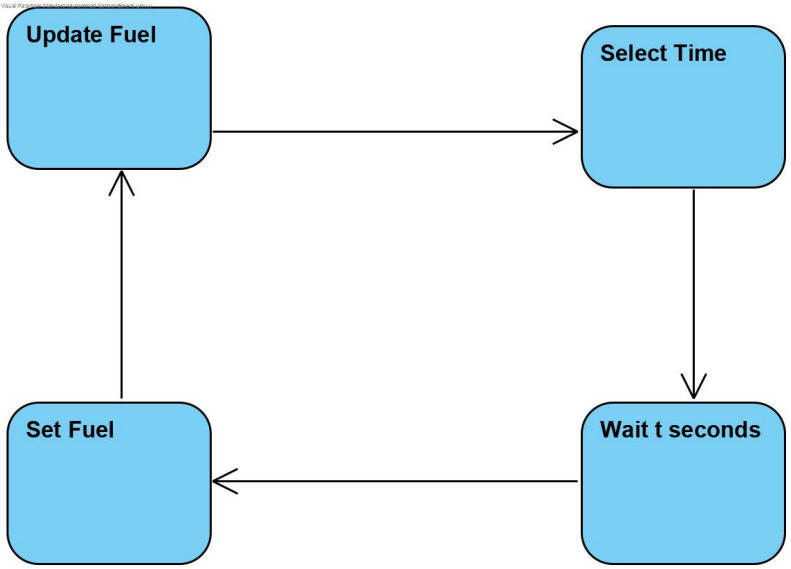


Figure 9

One of the workings that the system handles is updating the fuel of the spaceship. The fuel reduces a certain amount after every few minutes as the spaceship moves. When the updateFuel() method is called, the time to wait before dropping the fuel is decided by the system and then after waiting for that specific amount of time the fuel percentage is dropped by a pre decided amount and the whole process is repeated again.

4.2.2.6. Score Update

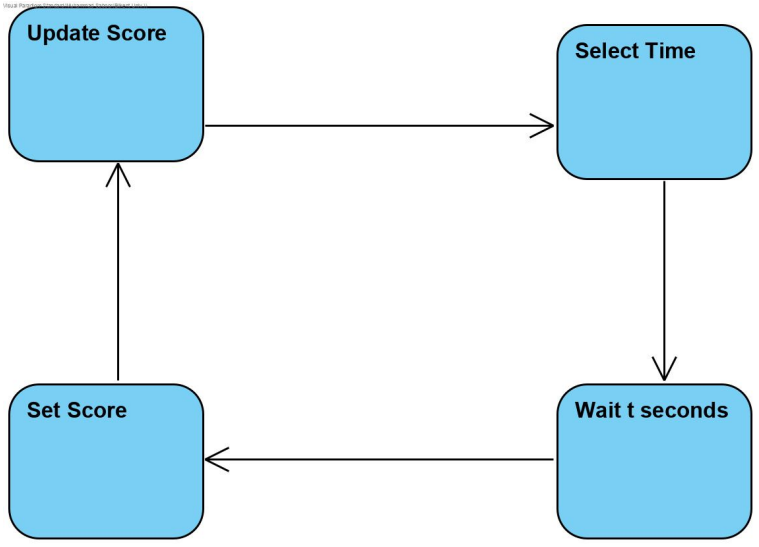


Figure 10

Updating the score is a very important feature of the game as this will allow the player to get the score according to the time spent inside the game. When the method updateScore() is called, the system is signalled to wait a few seconds before updating the score as to allow the player to play for some time and update the score after the set amount if the player is not dead. The time of wait and the amount of increase in the score is predetermined as to keep the player interested in the game.

4.2.2.7 Spaceship Collision

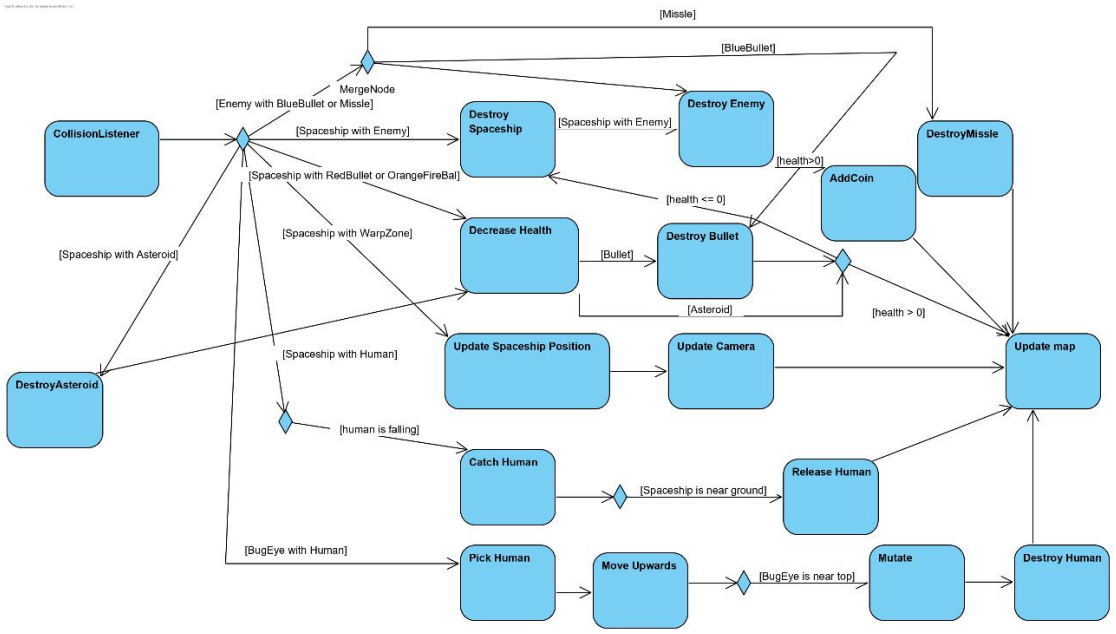


Figure 11

The description of the activity diagram is as follows:

1. The first action is collision between spaceship and player, in this case the system destroys both the spaceship and the enemy.
2. The second interaction is collision between spaceship and (enemy) bullet, in this case the system must decrement spaceship health and destroy the collided bullet (furthermore if the bullet decrements the spaceship's health to less than or equal to zero, then the system must destroy the spaceship).
3. The third activity is collision between spaceship and warp zone, the system handles this by updating the spaceship position and updating the camera position to follow the player accordingly.
4. The spaceship may also fire a bullet, in this case if a collision is detected between the enemy and the bullet the system must destroy both of them.
5. The spaceship may fire missiles as well and the missile after colliding with the enemy will be destroyed along with the enemy.
6. If a falling human collides with a spaceship the system must allow the spaceship to catch it and when it reaches the ground the spaceship automatically releases the human.
7. The bugeye (enemy class) may also collide and with a human, pick it and then move upwards, in that case if it reaches the top of the camera view the human bugeye mutates (into stronger enemy) and the human is destroyed.

8. If the bugeye is destroyed whilst carrying a human, that human will be dropped and the player spaceship will have to collide with the human to pick them up and then go near the ground in order to drop the human.
9. We will also have asteroids on the map and if a collision occurs the spaceship will take heavy damage while the asteroid will be destroyed.
10. The game includes health and fuel pickups, if the player collides with any of these items the player will have their stats restored and the pickup will disappear.
11. Upon death enemies will drop coins, however the player does not need to collide with these coins and the coins will disappear into the players inventory shortly after being dropped by enemies.

    For each of the aforementioned events the map and camera are updated.

4.2.3 State Diagrams

4.2.3.1 TitleScreen state diagram

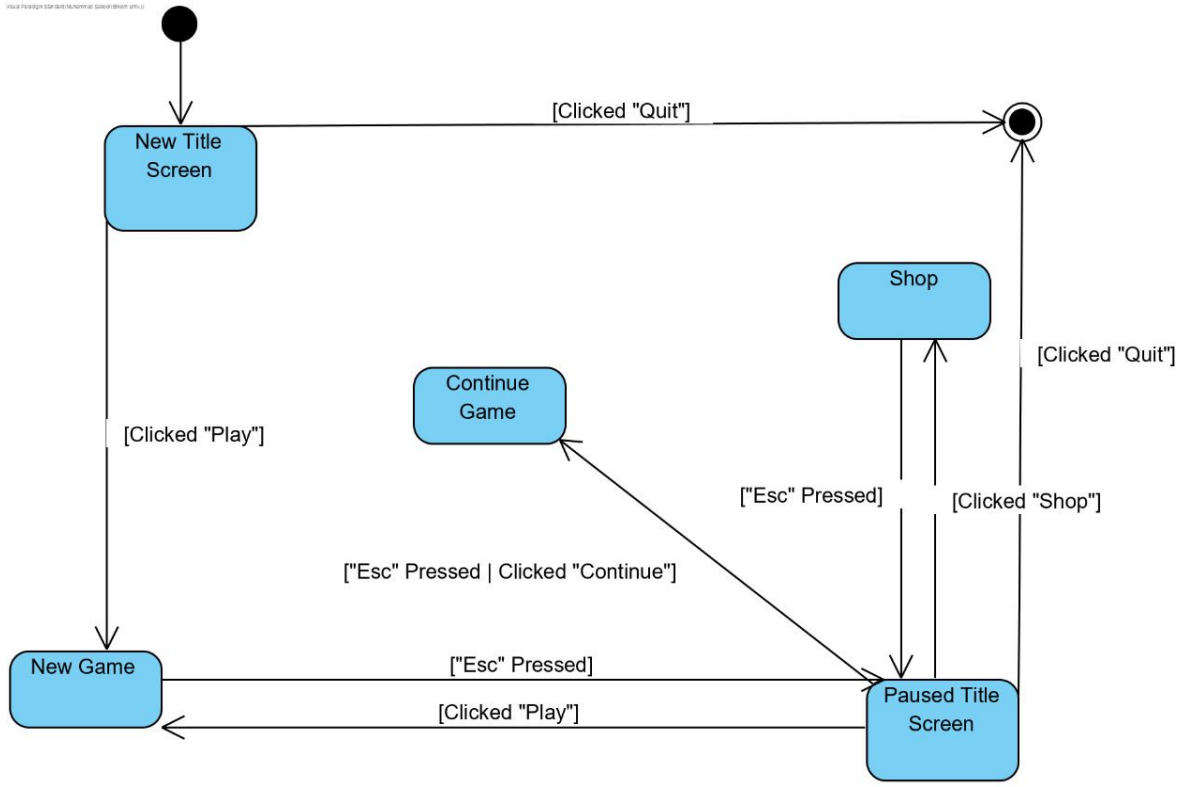


Figure 12

Upon startup the user will be taken to the New Title Screen and if the user clicks the button “Quit” they will exit the game. The second option here is to click “Play” after which a new game will be started, while playing the game if the user presses the “Esc” key they will be taken to the Paused Title Screen from here the user may select “Play” and start a new game or press “Shop” to be taken to the shop screen, or the user may press “Quit” and exit the game, otherwise the game will continue. In the shop screen the player may again press “Esc” to return to the Paused Title Screen.

4.2.3.2 Coin state diagram

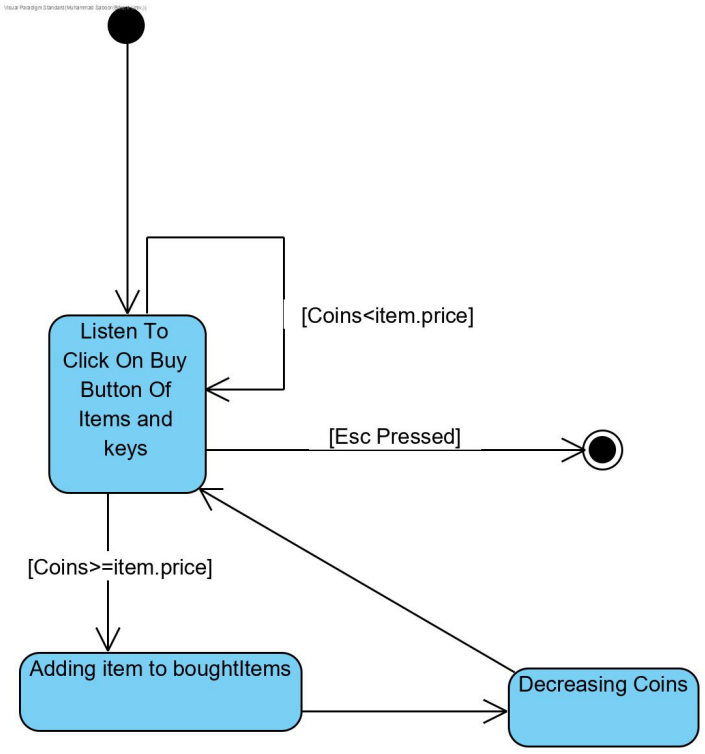


Figure 13

In the beginning the game will listen to the buttons pressed by the user, if the user pressed the button to buy an item and the amount of coins is less than the price the game will keep listening and the purchase will not be made. If the user has coins more than or equal to the price of the item, the item will be bought and the amount of coins will be decreased.

4.3 Object and Class model

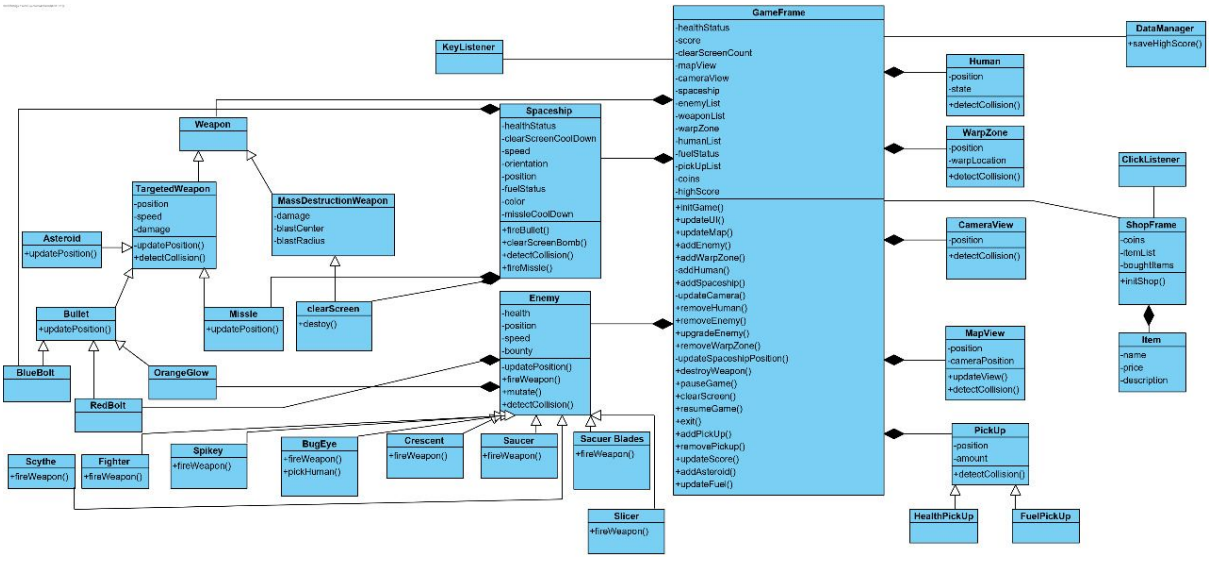


Figure 14 (Object and Class Diagram)

This diagram basically explains the whole class structure of any components participating in the game. As it can be seen, the main class that controls basically everything in the game is the GameFrame.

The GameFrame is the main class which will be instantiated when the game is opened. This class has two jobs, one is display everything on the screen and the other is to control the flow of the game and bridge the gap between all the other components of the game. GameFrame has many methods which help in initializing the many components we want in the game such as the spaceship, enemies, humans, asteroids, pickups etc. This class is also in charge of keeping the score of the player up to date and display it on the screen.

Spaceship is a class which is initialized at the start of the game and it can be controlled by the player. This class has some attributes such as health, fuel, orientation, position etc. These help the placement of the spaceship on the map and tracking the progress of the player and whether they crashed the plane or not. This class also has some methods such as fireBullet(), clearScreenBomb(), detectCollision() etc. These methods help in accomplishing the different tasks of the spaceship such as firing weaponry detecting whether the ship has collided with anything or not. This class can not exist without the game frame and has to be instantiated on top of it.

Enemy is a class from which all of the enemy types are inherited. It has a lot of methods and attributes which are common among all of the enemy types such as health, speed and position of the enemy the same as the spaceship. It also has fireWeapon(), mutate() and detectCollision() to complete the duties it will have during the game. This class is inherited by 8 enemy types of which 2 are boss types and 6 are normal enemies. These 8 enemies have a fireWeapon() for each of the type as the type of weapon they fire is different. The BugEye also has a pickHuman() method as it is the only one which can pickup humans. These classes are also not independent of the GameFrame and have to be instantiated on top of it.

Weapon is a class which is inherited by 2 different kinds of weapons in the game. These types are MassDestructionWeapon and TargetedWeapon. The MassDestructionWeapon can only be fired by the spaceship. This class contains the starting point of the attack and the radius of the attack and aims to destroy any enemy in that radius. Its method destroy() does exactly that. On the other hand, targetedWeapon can be fired by the enemy as well. The TargetedWeapon is inherited by Asteroid, Bullet and Missile and the Bullet is further inherited by BlueBolt, RedBolt and OrangeGlow. The Blue Bolt is fired by the player, the OrangeGlow and RedBolt are fired by the enemy. The Missile is an enemySeekerMissile and can only be fired by the spaceship. The asteroids are independent of any enemy or player as it will be spawned by the system to target the spaceship.

The human class is rather simple as the humans will only be instantiated on top of the frame and their position will be recorded to see where they are placed and their collisions will be recorded as we have to allow the BugEye to pick them up on contact.

The WarpZone is the class instantiated on the GameFrame and it constantly monitors its collisions to see whether the spaceship has collided with it to transport it to the final warpLocation. The WarpZone also has position attribute which will show the coords on which this WarpZone is located.

The Pickup class basically shows which pickups are available in the game. It is inherited by HealthPickup and FuelPickup and both of them have the position on which they both are at that moment and the amount of change they will bring to the amount of fuel or health the spaceship when they collide with the spaceship.

The MapView is the whole map of the game. It basically shows the place where everything on the map is and also shows where the camera is currently using its cameraPosition attribute.

The CameraView is the class which monitors which part of the Mapview should be shown on the screen and it updates these coords when the spaceship moves.

The ShopFrame is the class which is instantiated independent from the GameFrame and where the player can buy different upgrades for the Spaceship. It contains coins, itemList, boughtIems attributes.These help in comparing whether the player can afford the item that the player wants to buy.

We have a DataManager class which is required to save the current score of the player if it is greater than the previous highScore to save the new HighScore.

The ClickListener and the KeyListener are classes which listen to any activity on the class they have been attached to and are used to interact with the user of the game.

4.4 User Interface - Navigation Paths and Screen Mockups

Fig15. Start Menu

This is the menu initially presented to the user as the game starts, the player can choose to buy items, start the game or quit the game. Items include an upgrade to the shield of fuel tank.

Fig 16. Pause Menu

This is the menu presented to the user when the game is paused. Here too the player can choose to upgrade his current items, start the game or quit the game. Items include an upgrade to the shield of fuel tank.

Fig 17. Gameplay

This is an example of what the Game Frame will look when the game is played. There is a bar that depicts the current fuel at the top, this drops as time goes on. There is a map in the top center that represents the location of player, enemies, humans and the Boss(represented as the big red item, if any). Upon clicking the fire button the player shoots, and tries to hit the enemies, which die upon taking a certain amount of damage. There are a number of enemies on the screen and one can be observed reaching down for a human.

Player(Spaceship)

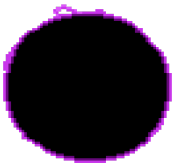
Bugeye(Enemy)

Fighter(Enemy)

Slicer(Enemy)

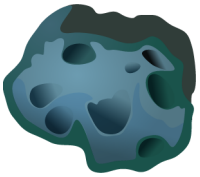
Spikey(Enemy)

Saucer(Enemy)



Portal

Boss(Map Location)



Meteor

Human