# PROG1300 Project Documentation For Space Shooters

Author:  Justin Sumrall
Date: December 5, 2020

## Table of Contents

1. Project Overview ... 1
2. Project Requirements ... 1
2.1. Use Case Diagram ... 1
2.1.1. Start Game ... 2
2.1.2. Move Ship ... 3
2.1.3. Fire Lasers ... 3
2.1.4. Player Hit ... 4
2.1.5. Enemy Hit ... 4
2.1.6. Game Ends ... 5
2.2. Activity Diagrams ... 5
2.2.1. Activity Diagram #1 ... 6
2.3. Derived Requirements ... 7
3. Design Plans ... 7
3.1. Class Diagram ... 8
3.2. Sequence Diagram ... 9
4. Implementation ... 9
4.1. Screen Snippets ... 9
4.2. Code Snippet #1 ... 12
4.3. Code Snippet #2 ... 13
5. Testing Paragraphs ... 13
5.1. Unit Testing ... 13
5.2. Integration Testing ... 17
Appendix A. Requirements Traceability ... 18

## List of Tables

Table 1: Bullet Collision Handling ... 12
Table 2: Bullet Movement and Removal ... 13

## List of Figures

Figure 1: Use Case Diagram of the flow of gameplay ... 2
Figure 2: Start Screen with Instructional Text ... 10

## 1. Project Overview

In this project I will be developing a 2D top-down perspective space shooter game. The player will control a spaceship, fire projectiles at enemies to collect points, and attempt to evade enemy projectiles. The game will be cleared after a certain number of enemy ships are defeated. After the game is over, either by destroying all enemies or by being defeated yourself, the player's final score will be displayed.

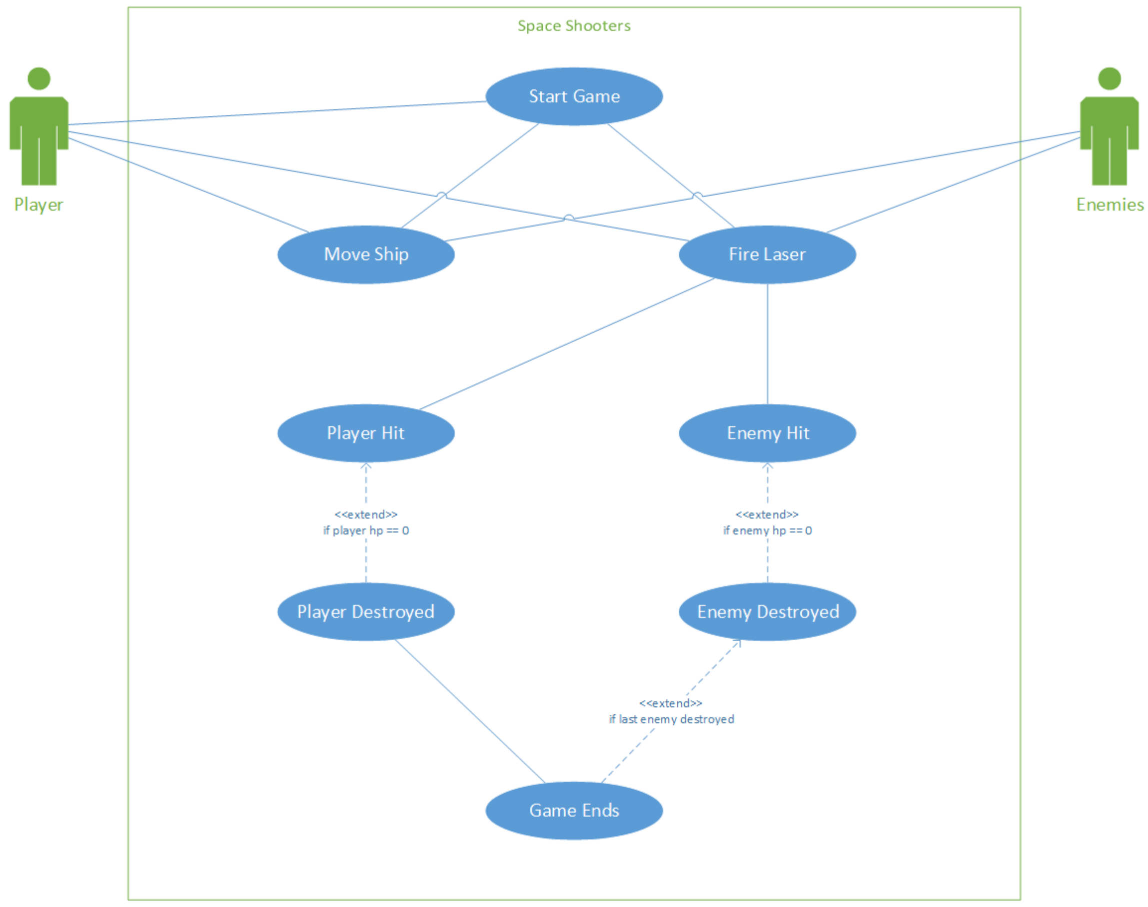
## 2. Project Requirements

- **The player's ship moves according to player input.**
- **The player's ship fires lasers to damage enemy ships according to player input.**
- **Enemy ships move and shoot at the player's ship automatically.**
- **Health of all ships are tracked, but health is not visible to player.**
- **After game ends, player's final score must be displayed.**

### 2.1. Use Case Diagram

From the following diagram you can see that lots of things are going on even though functionally the player only has two main options during gameplay to control their ship.

**Figure 1: Use Case Diagram of the flow of gameplay**



## 2.1.1. Start Game

- **Preconditions:** The game program is run.
- **Steps:**

  **1.** The game window opens, the canvas is black except for white text instructing the player of the controls for the game.

  **2.** The player presses the space bar once they are ready to begin playing.
- **Postconditions:** The game begins, the instructional text disappears, player ship appears and becomes controllable, and enemy ships begin to appear.

### 2.1.2. Move Ship

- **Preconditions:** Game has started.
- **Steps:**

  **1.** The player presses arrow keys to move their ship in the desired direction.

  **2.** The player attempts to use this movement to attempt to avoid enemy lasers,or position themselves for firing lasers at enemies.

- **Postconditions:** The player successfully dodges an enemy laser, fails to dodge an enemy laser, and/or positions themselves to fire a laser.

### 2.1.3. Fire Lasers

- **Preconditions:** Game has started.
- **Steps:**

  **1.** For each time the player presses the space bar, a laser projectile appears just above the current position of the player's ship.

  **2.** The player's laser projectile then moves upwards along the game canvas.

  **3.** At random intervals, a laser projectile may appear just below enemy ships.

  **4.** Enemy lasers then move downwards along the game canvas.

- **Postconditions:** Lasers move along the canvas until their positions either overlap with a ship or they move off the visible boundaries of the game window, whichever comes first.

### 2.1.4. Player Hit

- **Preconditions:** Enemy laser position overlaps with player ship position.
- **Steps:**

  **1.** The hp value of the laser decrements from 1 to 0, causing it to be removed from the canvas.

  **2.** The hp value of the player's ship decrements by 1.
- **Postconditions:** The game applies the updated hp value to the player ship object. If this results in the player ship's hp being reduced to zero then the player ship is destroyed.

### 2.1.5. Enemy Hit

- **Preconditions:** Player laser position overlaps with enemy ship position.
- **Steps:**

  **1.** The hp value of the laser decrements from 1 to 0, causing it to be removed from the canvas.

  **2.** The hp value of the enemy ship decrements by 1.
- **Postconditions:** The game applies the updated hp value to the enemy ship object. If this results in the enemy ship's hp being reduced to zero then the enemy ship is destroyed.

### 2.1.6. Game Ends

- **Preconditions:** As the player destroys enemy ships, they will accumulate score points. Once all enemies or the player’s ship is destroyed, then the game will end.
- **Steps:**

  **1.** Screen is cleared of all objects, player controls disabled.

  **2.** The total amount of points the player has earned will be displayed.

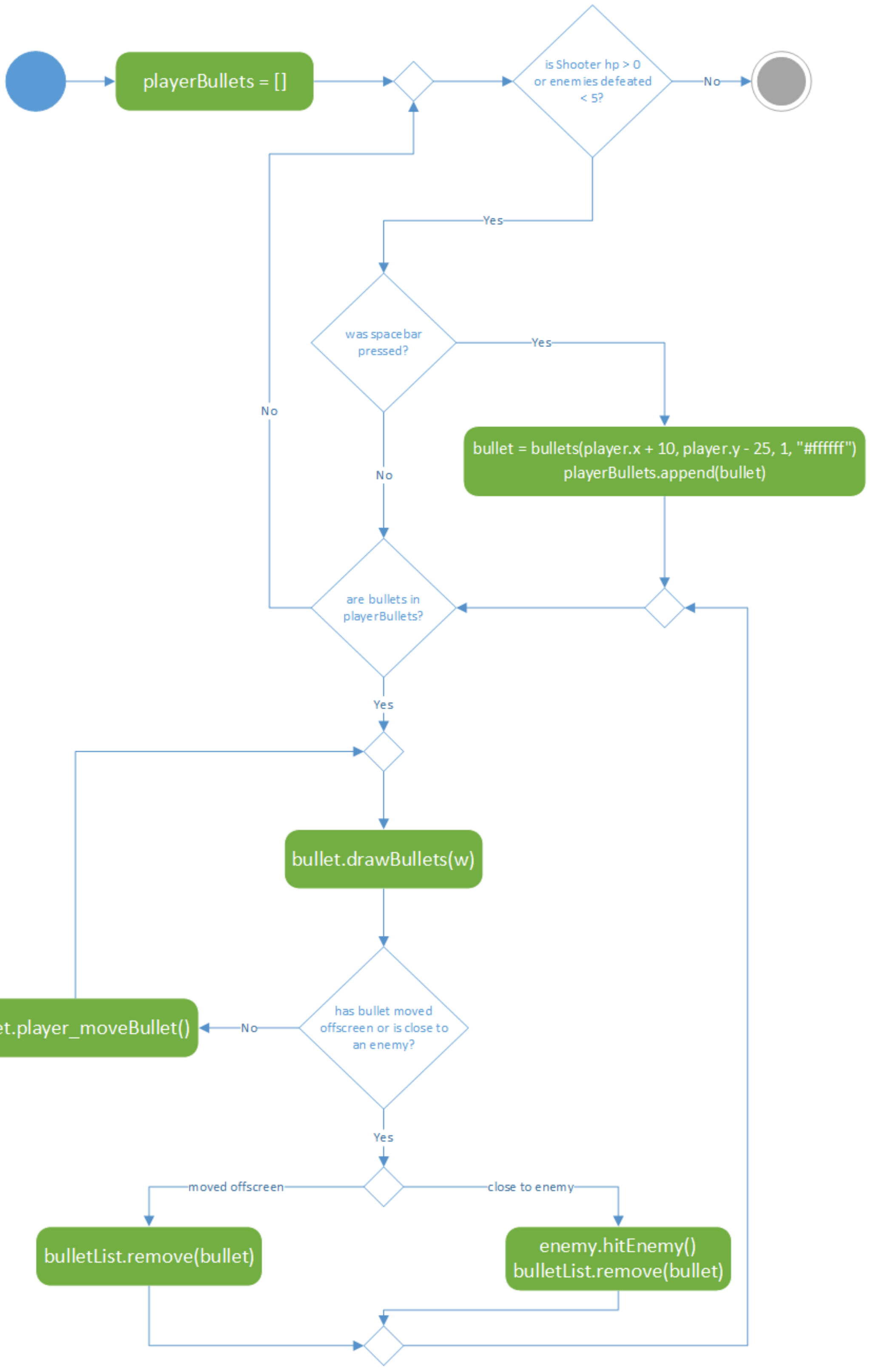
  **3.** Player presses Escape key to close game.

- **Postconditions:** Game window closes. Game program terminates.

## 2.2. Activity Diagrams

Currently, I have here an activity diagram to show the lifecycle of the bullet objects created by the player. If more diagrams are needed they will be added and this section will be updated.

## 2.2.1. Activity Diagram #1


playerBullets = []
is Shooter hp > 0 or enemies defeated < 5?
No
Yes
was spacebar pressed?
Yes
No
bullet = bullets(player.x + 10, player.y - 25, 1, "#ffffff")
playerBullets.append(bullet)
No
are bullets in playerBullets?
Yes
bullet.drawBullets(w)
has bullet moved offscreen or is close to an enemy?
No
bullet.player_moveBullet()
Yes
moved offscreen
close to enemy
bulletList.remove(bullet)
enemy.hitEnemy()
bulletList.remove(bullet)

## 2.3. Derived Requirements

### 2.3.1. Enemy Ship Movement

Enemy ships must move and fire lasers with no regard for player input. They must be controlled algorithmically by the game's code. The main program must accurately decrement the health of enemy ships as they are hit by lasers, and destroyed once they have zero health remaining.

### 2.3.2. GUI Window

The main program needs to open a GUI window in order to display the game elements, such as the player ship, enemy ships, and laser objects.

### 2.3.3. Laser Collision

Ships need to be able to detect if a laser has gotten close enough to them to trigger their health decrement methods.

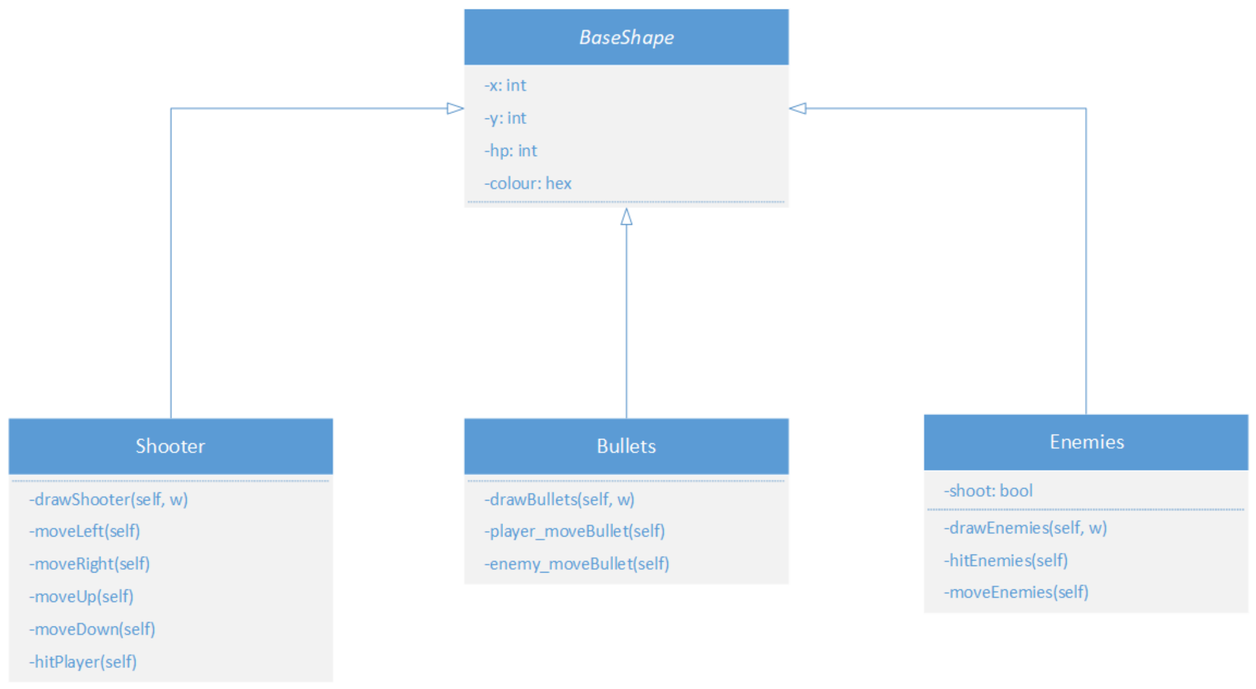
### 2.3.4. Laser Removal

Laser objects must be removed from the screen after hitting a ship or going offscreen.

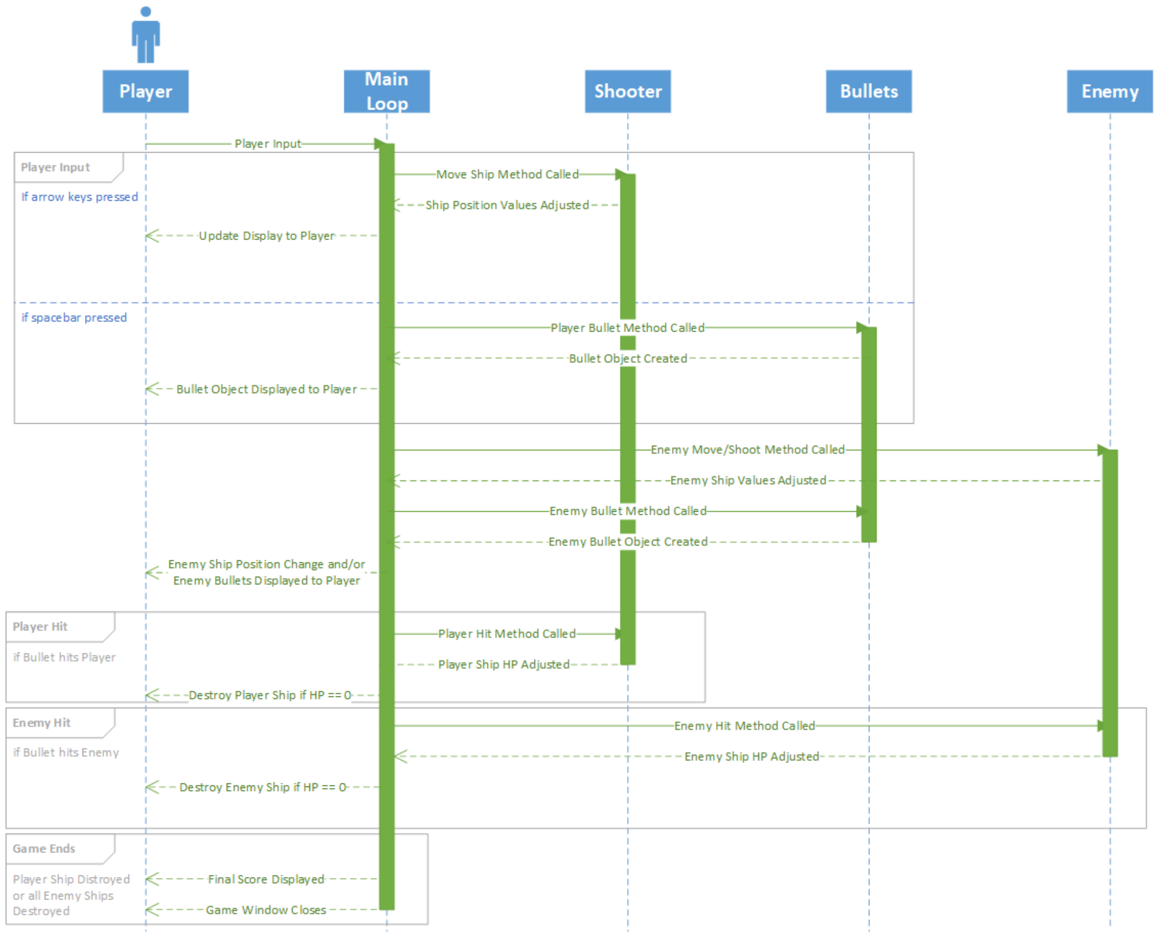
## 3. Design Plans

The overall design of the game is fairly simple. Dodge enemy lasers and fire your own lasers at the enemies. The game will use a base shape class with three child classes that inherit from it: Shooter (The player's ship), Bullets (the laser objects), and Enemies.
In the following diagrams I will show class and sequence diagrams to explain the relationship between these classes.

## 3.1. Class Diagram



## 3.2. Sequence Diagram


Player
Main Loop
Shooter
Bullets
Enemy
Player Input
Player Input
If arrow keys pressed
Move Ship Method Called
Ship Position Values Adjusted
Update Display to Player
if spacebar pressed
Player Bullet Method Called
Bullet Object Created
Bullet Object Displayed to Player
Enemy Move/Shoot Method Called
Enemy Ship Values Adjusted
Enemy Bullet Method Called
Enemy Bullet Object Created
Enemy Ship Position Change and/or Enemy Bullets Displayed to Player
Player Hit
if Bullet hits Player
Player Hit Method Called
Player Ship HP Adjusted
Destroy Player Ship if HP == 0
Enemy Hit
if Bullet hits Enemy
Enemy Hit Method Called
Enemy Ship HP Adjusted
Destroy Enemy Ship if HP == 0
Game Ends
Player Ship Distroyed or all Enemy Ships Destroyed
Final Score Displayed
Game Window Closes

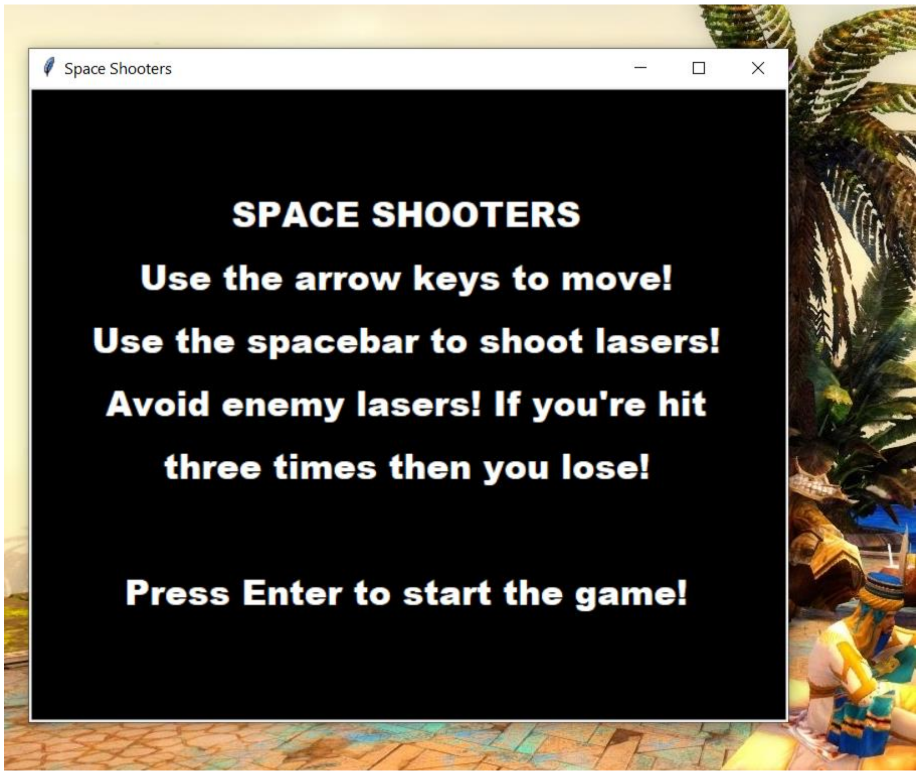
## 4. Implementation

The code from the PROG1700 2D Game assignment was taken and modified so that all game elements were contained within the game window and not split between the game window and the python console.

### 4.1. Screen Snippets

The following three snippets show the main features of the game working, the start screen with instructional text being displayed, the actual game running, and then the final score screen which now stays open until the user presses the escape key.

**Figure 2: Start Screen with Instructional Text**


Space Shooters

SPACE SHOOTERS

Use the arrow keys to move!

Use the spacebar to shoot lasers!

Avoid enemy lasers! If you're hit three times then you lose!

Press Enter to start the game!

**Figure 3: Main game program loop running**

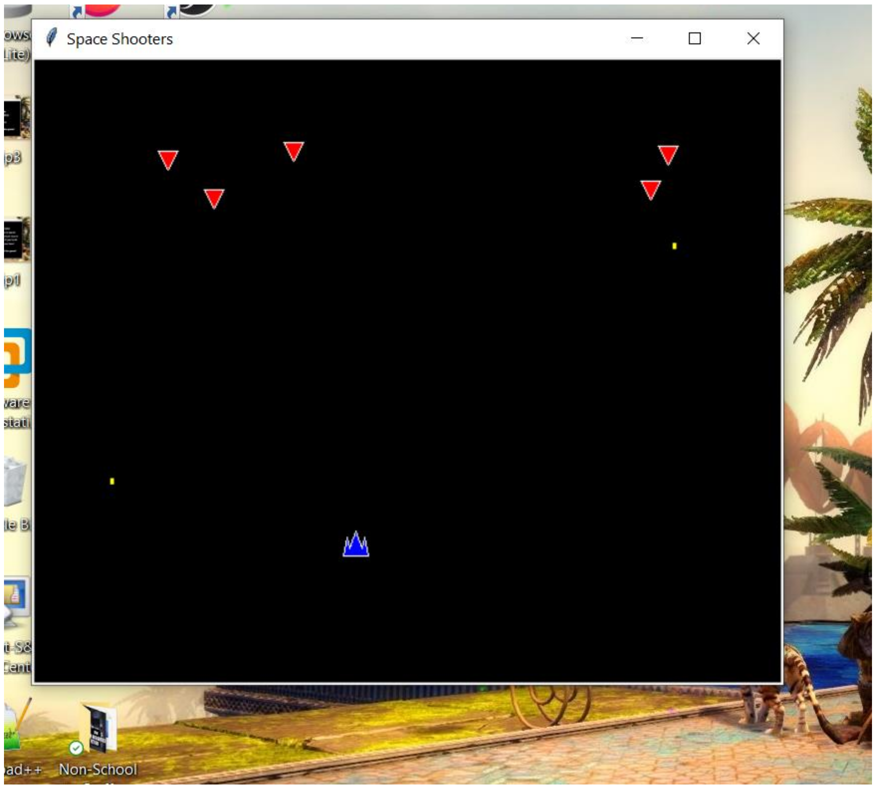
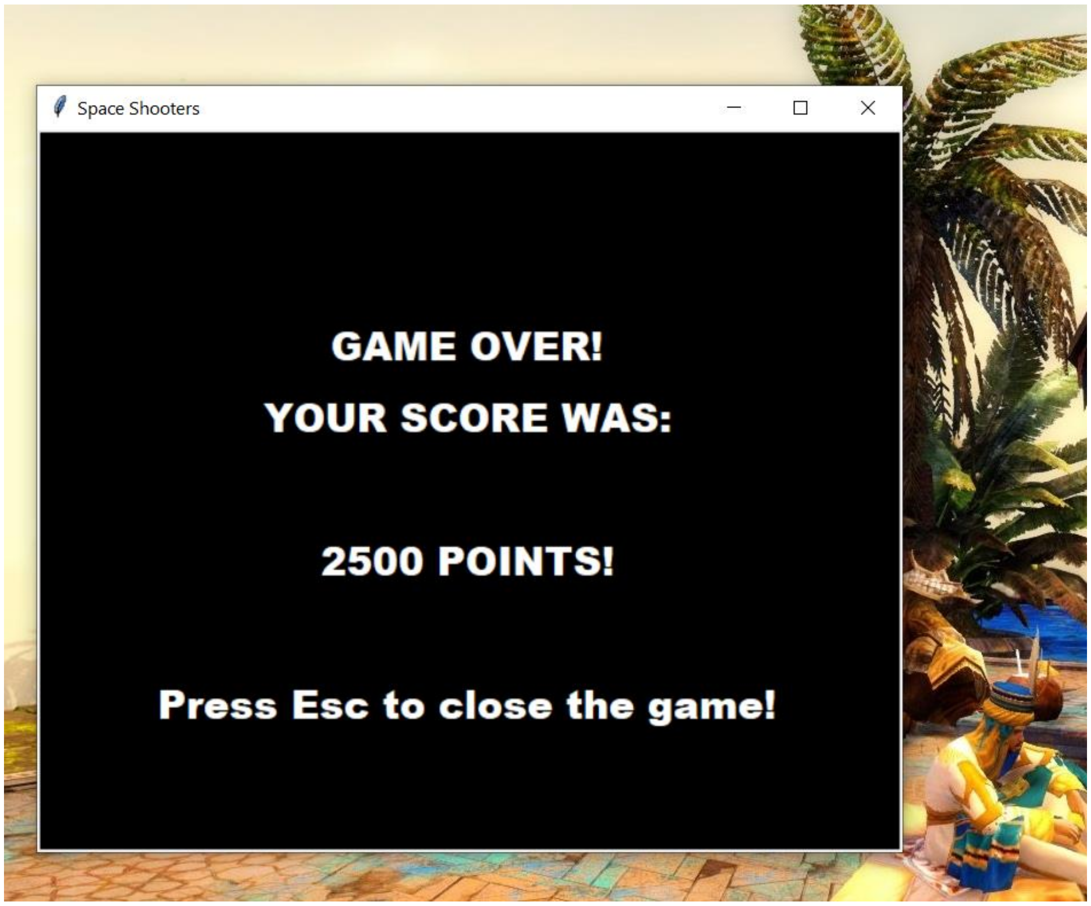


**Figure 4: Final score screen with displayed score and instructions to close window**



GAME OVER!

YOUR SCORE WAS:

2500 POINTS!

Press Esc to close the game!

## 4.2. Code Snippet #1

This section of code handles collision detection for both the player’s lasers vs enemy ships and for enemy lasers vs the player’s ship. Once the absolute value of the distance between a player’s laser object and an enemy ship is less than 15 pixels on both the x and y axes then the bullet’s hp is decremented and the health decrement function is also called on that enemy ship. Similiarly, if the absolute distance between an enemy laser and the player’s ship is less than 20 on both the x and y axes then the enemy laser’s hp is decremented and the health decrement function is called on the player’s ship.

**Table 1: Bullet Collision Handling**

```
# This loop handles player bullet vs enemy collision
for b in playerBullets:
    for e in enemyList:
        if e.enemyCollide(b):
            b.hp -= 1
            e.hitEnemies()
# This loop handles enemy bullet vs player collision
for b in enemyBullets:
    if player.playerCollide(b):
        b.hp -= 1
        player.hitPlayer()

def playerCollide(self, other):
    '''Method for collision detection against the player.'''
    if (abs(other.x - self.x) < 20) and (abs(other.y - self.y) < 20):
        return True
    else:
        return False

def enemyCollide(self, other):
    '''Method for collision detection against an enemy.'''
    if (abs(other.x - self.x) < 15) and (abs(other.y - self.y) < 15):
        return True
    else:
        return False
```

### 4.3. Code Snippet #2

This section of code handles drawing player/enemy lasers and removing them once either their health is decremented by collision detection or if they go off screen by either their y position going below zero (in the case of player lasers) or their x position going above 500 (in the case of enemy lasers), as long as both of these conditions are false the lasers continue to move across the screen instead.

**Table 2: Bullet Movement and Removal**

```
# For each bullet the player fires this loop has it move up until it
hits an enemy or goes off screen.
for b in playerBullets:
    b.drawBullets(w)
    if b.bulletCheck():
        playerBullets.remove(b)
    else:
        b.player_moveBullet()

# For each bullet an enemy fires this loop has it move down until it
hits the player or goes off screen.
for b in enemyBullets:
    b.drawBullets(w)
    if b.bulletCheck():
        enemyBullets.remove(b)
    else:
        b.enemy_moveBullet()

def bulletCheck(self):
    '''Detects if bullet has hit a ship or gone offscreen.'''
    if self.hp <= 0 or self.y < 0 or self.y > 500:
        return True
    else:
        return False
```

## 5. Testing Paragraphs

.

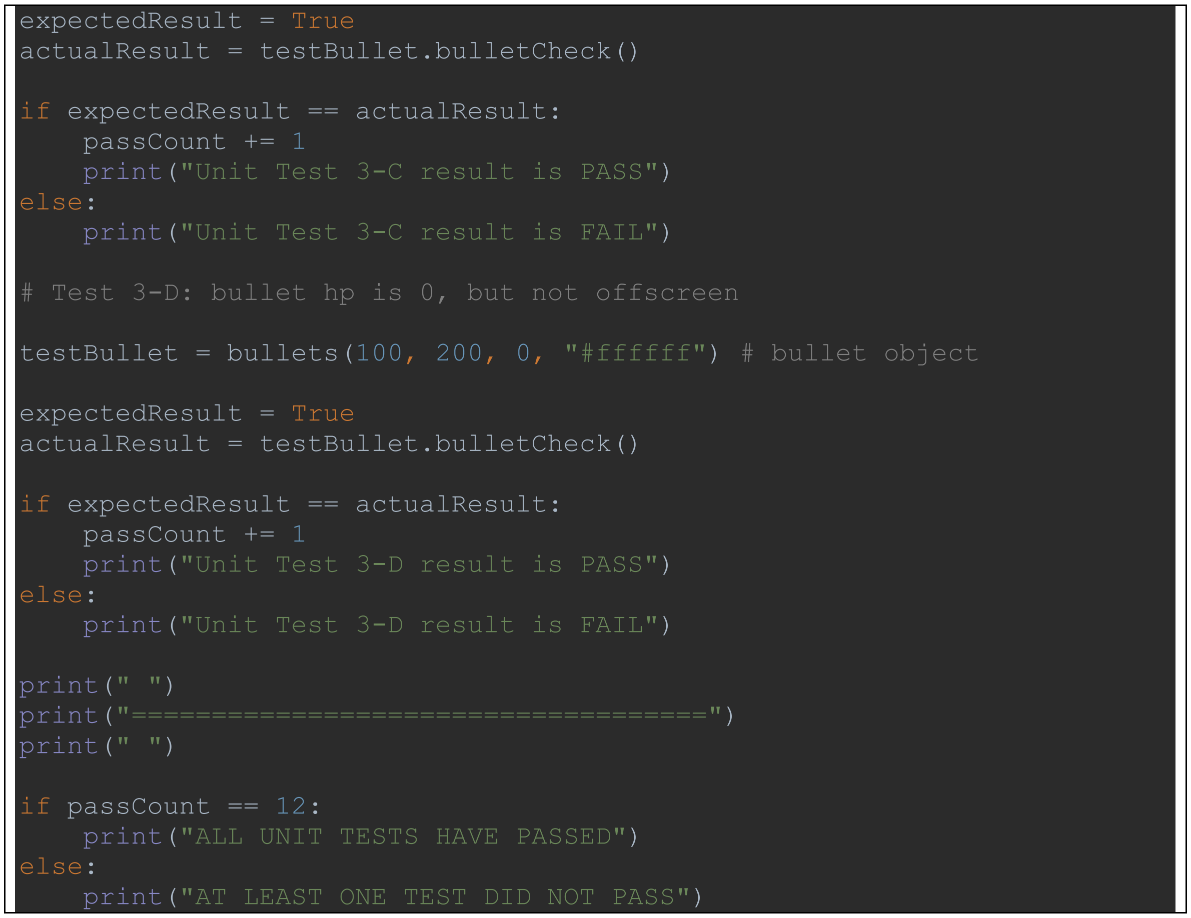
### 5.1. Unit Testing

Three above three methods: enemyCollide(), playerCollide(), and bulletCheck() were each unit tested to ensure that the methods function properly. In the table below is the entire Unit Test code for these methods, in the video demonstration attached with this document I show that all of these unit tests are successful.

**Table 3: Unit Testing Code**

```python
# unit_test.py

from Enemies import enemies
from Shooter import shooter
from Bullets import bullets

passCount = 0

print(" ")
print("====================================")
print(" ")

# Unit test 1, bullet collision vs player

# Test 1-A: bullet far from player

testPlayer = shooter(50, 50, 1, "#ffffff") # player object
testBullet = bullets(200, 200, 1, "#ffffff") # bullet object

expectedResult = False
actualResult = testPlayer.playerCollide(testBullet)

if expectedResult == actualResult:
    passCount += 1
    print("Unit Test 1-A result is PASS")
else:
    print("Unit Test 1-A result is FAIL")

# Test 1-B: bullet close on one axis, far on the other

testPlayer = shooter(50, 50, 1, "#ffffff") # player object
testBullet = bullets(55, 200, 1, "#ffffff") # bullet object

expectedResult = False
actualResult = testPlayer.playerCollide(testBullet)

if expectedResult == actualResult:
    passCount += 1
    print("Unit Test 1-B result is PASS")
else:
    print("Unit Test 1-B result is FAIL")

# Test 1-C: bullet close to player

testPlayer = shooter(50, 50, 1, "#ffffff") # player object
testBullet = bullets(55, 40, 1, "#ffffff") # bullet object

expectedResult = True
actualResult = testPlayer.playerCollide(testBullet)

if expectedResult == actualResult:
    passCount += 1
    print("Unit Test 1-C result is PASS")
else:
    print("Unit Test 1-C result is FAIL")
```

```
# Test 1-D: bullet is just outside collision range

testPlayer = shooter(50, 50, 1, "#ffffff") # player object
testBullet = bullets(70, 30, 1, "#ffffff") # bullet object

expectedResult = False
actualResult = testPlayer.playerCollide(testBullet)

if expectedResult == actualResult:
    passCount += 1
    print("Unit Test 1-D result is PASS")
else:
    print("Unit Test 1-D result is FAIL")

print(" ")
print("====================================")
print(" ")

# Unit test 2, bullet collision vs enemy

# Test 2-A: bullet far from enemy

testEnemy = enemies(50, 50, 1, "#ffffff", False) # enemy object
testBullet = bullets(200, 200, 1, "#ffffff") # bullet object

expectedResult = False
actualResult = testEnemy.enemyCollide(testBullet)

if expectedResult == actualResult:
    passCount += 1
    print("Unit Test 2-A result is PASS")
else:
    print("Unit Test 2-A result is FAIL")

# Test 2-B: bullet close on one axis, far on the other

testEnemt = enemies(50, 50, 1, "#ffffff", False) # enemy object
testBullet = bullets(55, 200, 1, "#ffffff") # bullet object

expectedResult = False
actualResult = testEnemy.enemyCollide(testBullet)

if expectedResult == actualResult:
    passCount += 1
    print("Unit Test 2-B result is PASS")
else:
    print("Unit Test 2-B result is FAIL")

# Test 2-C: bullet close to enemy

testEnemy = enemies(50, 50, 1, "#ffffff", False) # enemy object
testBullet = bullets(55, 40, 1, "#ffffff") # bullet object

expectedResult = True
actualResult = testEnemy.enemyCollide(testBullet)

if expectedResult == actualResult:
    passCount += 1
    print("Unit Test 2-C result is PASS")
else:
    print("Unit Test 2-C result is FAIL")

# Test 2-D: bullet is just outside collision range

testEnemy = enemies(50, 50, 1, "#ffffff", False) # enemy object
testBullet = bullets(65, 35, 1, "#ffffff") # bullet object

expectedResult = False
actualResult = testEnemy.enemyCollide(testBullet)

if expectedResult == actualResult:
    passCount += 1
    print("Unit Test 2-D result is PASS")
else:
    print("Unit Test 2-D result is FAIL")

print(" ")
print("====================================")
print(" ")

# Unit Test 3, detecting if bullet has zero hp or has gone offscreen so
it can be flagged for removal
# Canvas size for this test is assumed to be 600 width and 500 height

# Test 3-A: bullet hp > 0, not offscreen

testBullet = bullets(100, 200, 1, "#ffffff") # bullet object

expectedResult = False
actualResult = testBullet.bulletCheck()

if expectedResult == actualResult:
    passCount += 1
    print("Unit Test 3-A result is PASS")
else:
    print("Unit Test 3-A result is FAIL")

# Test 3-B: bullet hp > 0, bullet is below canvas boundary

testBullet = bullets(100, 501, 1, "#ffffff") # bullet object

expectedResult = True
actualResult = testBullet.bulletCheck()

if expectedResult == actualResult:
    passCount += 1
    print("Unit Test 3-B result is PASS")
else:
    print("Unit Test 3-B result is FAIL")

# Test 3-C: bullet hp > 0, bullet is above canvas boundary

testBullet = bullets(100, -1, 1, "#ffffff") # bullet object
```



```
expectedResult = True
actualResult = testBullet.bulletCheck()

if expectedResult == actualResult:
    passCount += 1
    print("Unit Test 3-C result is PASS")
else:
    print("Unit Test 3-C result is FAIL")

# Test 3-D: bullet hp is 0, but not offscreen

testBullet = bullets(100, 200, 0, "#ffffff") # bullet object

expectedResult = True
actualResult = testBullet.bulletCheck()

if expectedResult == actualResult:
    passCount += 1
    print("Unit Test 3-D result is PASS")
else:
    print("Unit Test 3-D result is FAIL")

print(" ")
print("====================================")
print(" ")

if passCount == 12:
    print("ALL UNIT TESTS HAVE PASSED")
else:
    print("AT LEAST ONE TEST DID NOT PASS")
```

## 5.2. Integration Testing

For the Integration Testing I have included within the attached video file a demonstration of the game in its entirety running properly.

In my demonstration I go through all of the use cases in the Use Case Diagram within this document to prove that all requirements of the project have been met.

## Appendix A. Requirements Traceability

Below is an index of all the project requirements, what paragraph subsections they were defined in, and which paragraph subsections these requirements were tested in to prove that the defined requirements were met successfully.

| Paragraph Defined | Requirements Text | Paragraph Tested |
|---|---|---|
| 2.1.2. | The player's ship moves according to player input. | 5.2. |
| 2.1.3. | The player's ship fires lasers to damage enemy ships according to player input. | 5.2. |
| 2.3.1. | Enemy ships move and shoot at the player's ship automatically. | 5.2. |
| 2.3.1. | Enemy health tracked, but not visible to player. | 5.2. |
| 2.1.6. | After game ends, player's final score must be displayed. | 5.2. |
| 2.3.2. | GUI window opens when program starts. | 5.2. |
| 2.1.6. | Press escape to close game window once on final score screen. | 5.2. |
| 2.1.3., 2.1.4, 2.1.5., 2.3.4. | Laser objects must be removed from screen after hitting a ship or going offscreen. | 5.1., 5.2. |
| 2.3.3. | Ships need to detect if a laser has struck them. | 5.1., 5.2. |