Chaudhary Samir Zafar

AEE363

@00546317

Assignment Description

I have developed a game using JavaFX. I have implemented a series of design patterns into this game, and I will be breaking down the code and explaining why I used certain patterns.

DESIGN PATTERNS ASSIGNMENT

Chaudhary Samir Zafar | AEE363 | @00546317

Table of Contents

[UML Diagram](#_Toc68220629)

[Project Breakdown](#_Toc68220630)

[Game ideation](#_Toc68220631)

[Design Patterns choices](#_Toc68220632)

[Design Patterns Justification](#_Toc68220633)

[Factory – Justification](#_Toc68220634)

[Factory – UML](#_Toc68220635)

[Factory – Code implementation](#_Toc68220636)

[Singleton – Justification](#_Toc68220637)

[Singleton – UML](#_Toc68220638)

[Singleton – Code implementation](#_Toc68220639)

[Object pool – Justification](#_Toc68220640)

[Object pool - UML](#_Toc68220641)

[Object pool – code implementation](#_Toc68220642)

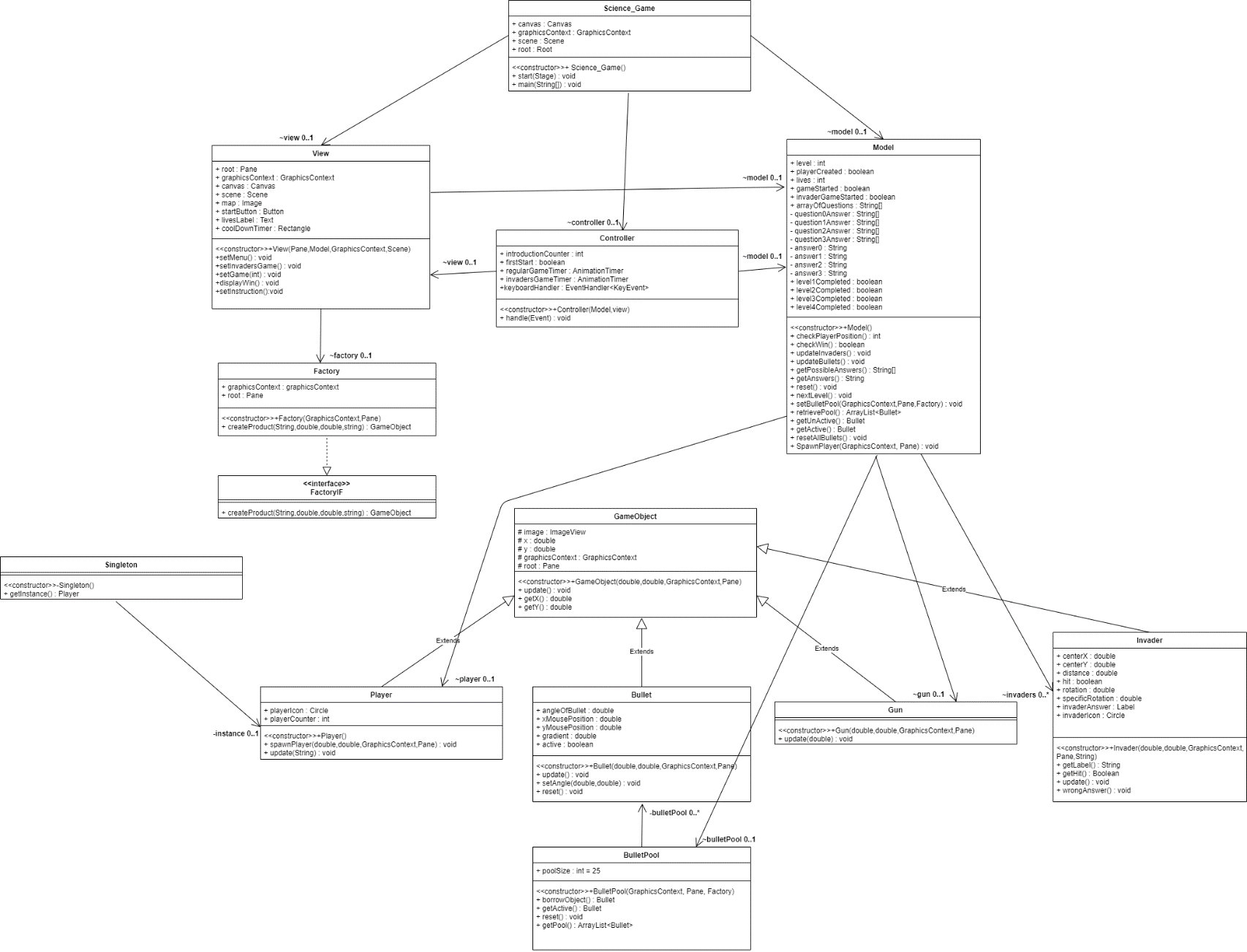
[MVC – Justification](#_Toc68220643)

[MVC – UML](#_Toc68220644)

[Evaluation](#_Toc68220645)

[References](#_Toc68220646)

# UML Diagram



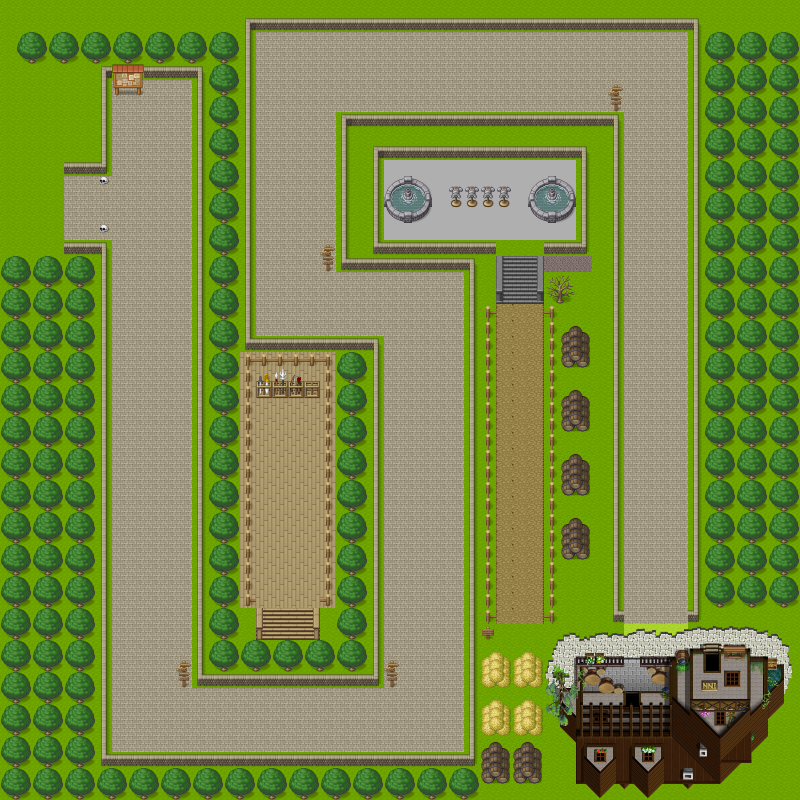
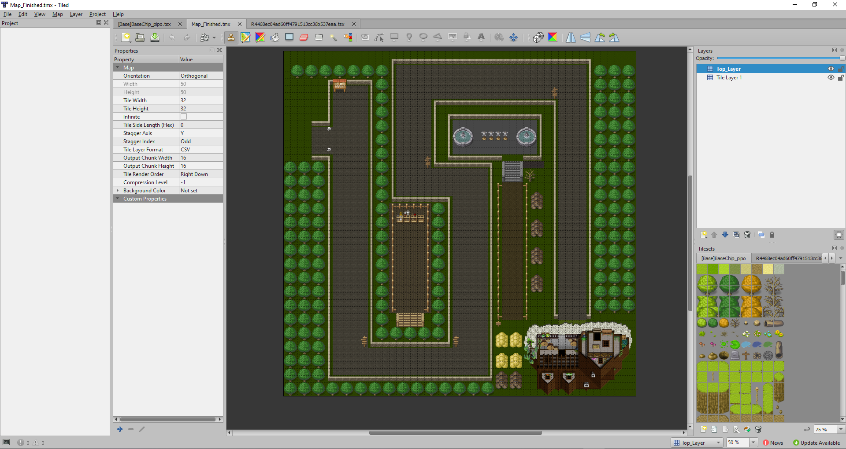
# Project Breakdown

### Game ideation

The development of a game takes a lot of thinking and planning. I took a similar approach when I was developing my game. The basic ideology of my game is you reach a checkpoint, and you face some invaders. To pass the level you have to shoot the invader labelled with the correct answer relative to a question being asked. The questions that are being asked are relative to science and appropriate for primary school children.

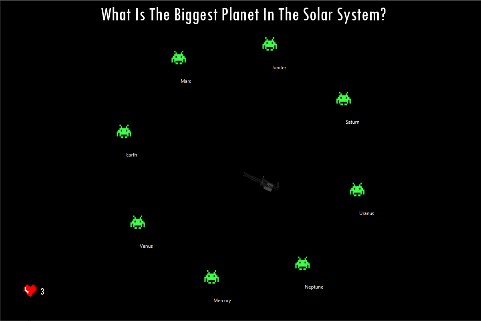
Here is an outlook of the map the player has to travel through and reach each checkpoint :

I created this map myself using software called Tiled.



* Each checkpoint is highlighted in red.
* The start is highlighted in blue.
* The end destination is highlighted in black.

Here is an outlook of the challenge a player will face at each checkpoint :



Question being asked – What is the biggest planet in the solar system?

If the player shoots at an invader that is not labelled Jupiter then the invader’s appearance will change so they know it is wrong and will not shoot that invader once again and a life will be deducted. If the player reaches 0 lives then the game will reset.

If the player shoots the correct answers the player will be spawned back onto the map ahead of the checkpoint.

### Design Patterns choices

The design patterns I chose to use for this game were :

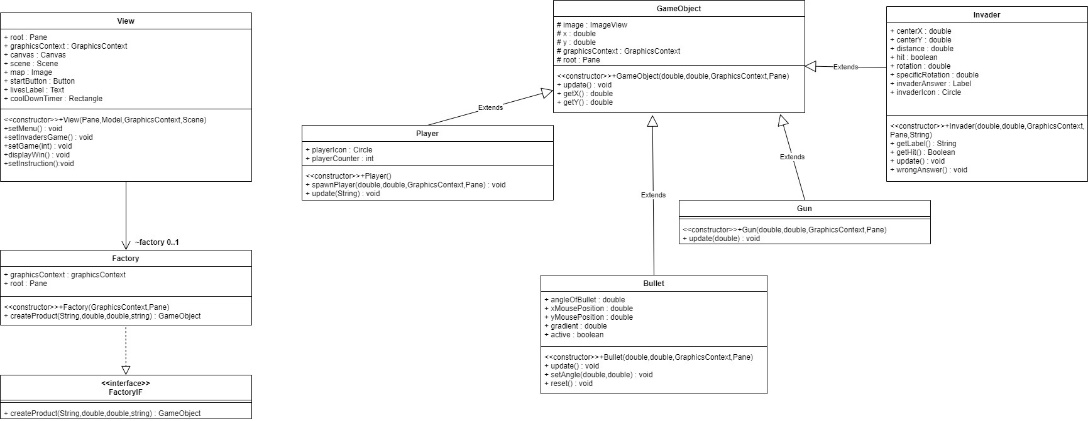
* Factory, Singleton, Object Pool, and MVC (Model, View, Controller)

### Design Patterns Justification

#### Factory – Justification

Factory synopsis – A class that can create instances of other classes without depending on them. The reusable class delegates the creation to another object and accesses it via a common interface.

##### Factory – UML



Factory is a creational pattern, and this is one of the main reasons I decided to implement this particular pattern. Throughout the whole game, the view needs to constantly produce different types of items onto the screen, and I would need new instances of particular classes to do this. The factory class allows the view to interact with many different types of objects using only one class.

##### Factory – Code implementation

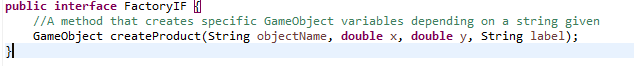
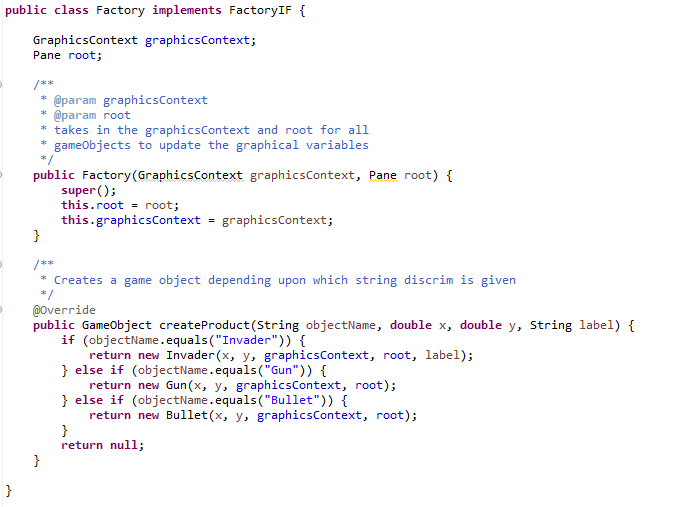


Figure 1 shows the interface that I have included in the factory pattern. The interface includes an abstract method. The method has no implementation. The implementation is done in the Factory class (Figure 2) which implements FactoryIF from figure 1. A class that is required to create arbitrary objects should not need to know the actual classes created therefore I have made the classes inherit from a common source. The source being the GameObject class in this case.

The method implementation is based upon the logic of whatever string value is passed in the parameters then return that type of object.

**Figure 1**

**Figure 2**

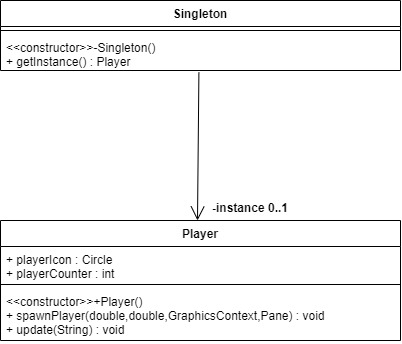


#### Singleton – Justification

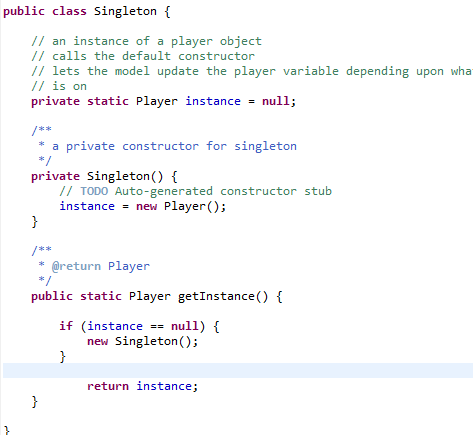
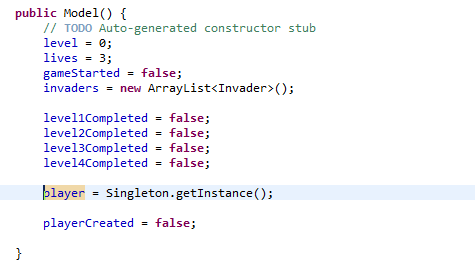
Singleton synopsis – This pattern only allows the creation of one instance of a class.

I identified the opportunity of using the singleton pattern when I looked further into the player object. Previously every time I was dying or passing a particular stage in the game I would create a new player and destroy the old one. However, upon refactoring my code I identified I could make use of the singleton pattern. Singleton is a creational pattern, and it would create my player instance solely once and I would use that particular instance throughout my whole game.

##### Singleton – UML

The singleton class consists of a private constructor and a getInstance() method that returns a static variable that holds a reference to an instance of a Player object.

##### Singleton – Code implementation



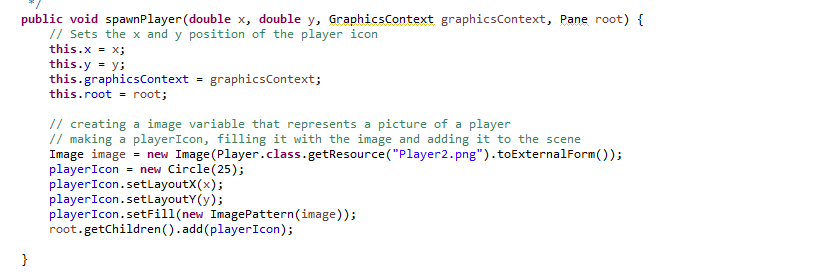
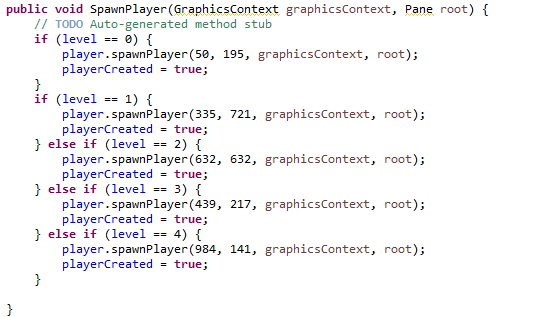
**Figure 2**

**Figure 1**

Figure 1 is the singleton class I have created. I have created a private constructor which initialised the instance variable. To create a Player instance, the static method of getInstance() is called from the external class. If the instance is equal to null then a new instance will be created by calling the constructor. Once this Player instance is created then no more instances of the class can be created.

Figure 2 is the constructor of my model class. I have declared a variable of type Player in my model class to keep track of the player positioning and deal with logic such as checking the current positioning of the player. If the player is crossing a checkpoint then trigger the invaders game. The model constructor is called once and that is at the start of the game, so this reinforces that only one Instance of Player is used.

However, using the singleton pattern I ran into a lot of technical issues regarding the player positioning after they complete levels. I improvised and created a method in the Player class and the model class to handle the player positioning depending upon the level the user is on.



**Figure 1**

**Figure 2**

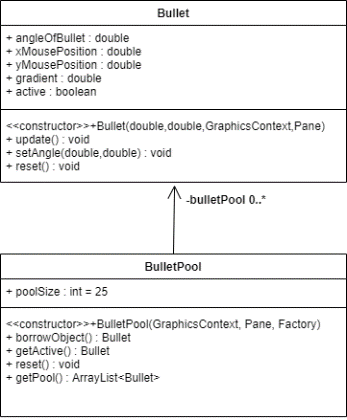
Figure one is a method from the model class. The spawnPlayer method (Figure 2) from the player instance is called and that spawns the player in the view. Every time the invaders game starts the player is cleared from the screen hence why I have added it to the root again, but the same instance is used throughout, and for debugging purposes, I used a static counter in the player class to see if the number ever exceeded 1 and it did not.

#### Object pool – Justification

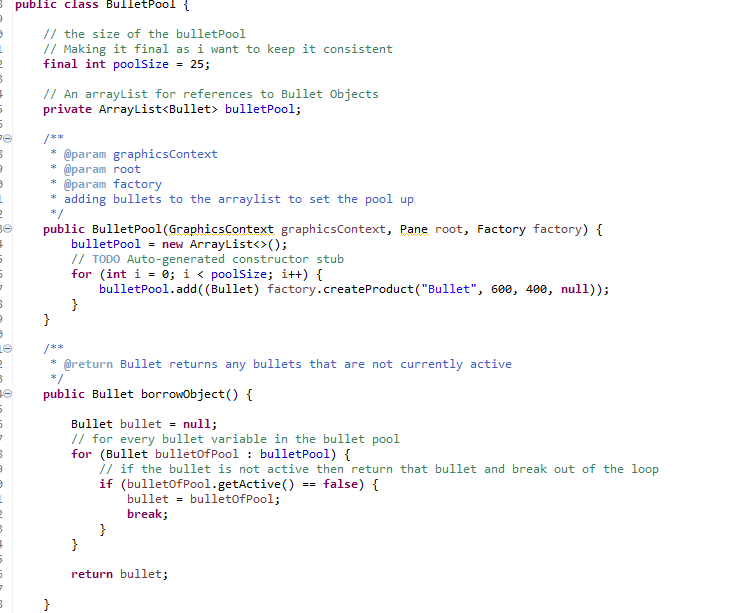
Object pool synopsis – when objects are expensive to create or limited in number manage their reuse.

I identified the opportunity to use the object pool design pattern when I was researching the best patterns to use for games. I have referenced the website I used to research potential patterns for games on my references page. I learned it was more efficient to reuse bullets instead of destroying them, so I decided to implement this pattern.

##### Object pool - UML

I created a final variable that declared the size of the pool. Through trial and error, I learned 25 was the best value to go for. I created an ArrayList of references to Bullet instances. Then the borrowObject() method would allow an external class to take a bullet instance. The getActive() method was needed to update bullets that are currently active, active in the respect that they have been shot out of the gun and their positioning needs to be updated constantly. The reset() method resets all of the bullets in the ArrayList.

##### Object pool – code implementation

This figure is a snip of my BulletPool class. When the class is created then an ArrayList of references to Bullet instances is created. I have created a variable in the Bullet class which tracks if the bullet is active or not. In the BorrowObject() method I iterate through the references of the ArrayList and once I find a Bullet instance that is not currently active then I pull that out of the pool.

In the bullet class, I have set a bullet to unactive when it leaves the screen, meaning the bullet can now be shot again by the gun.

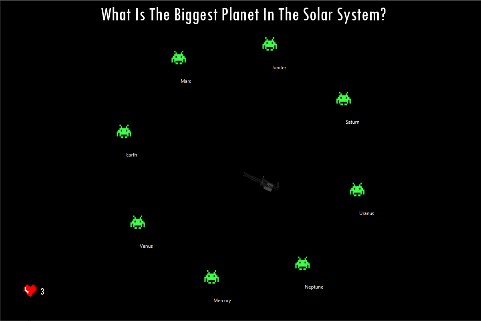
#### MVC – Justification

Model-View-Controller synopsis – (MVC) pattern separates the modeling of the domain, the presentation, and the actions based on user input into three separate classes.

I decided to use MVC due to the familiarity I already have with it. I have used this structural pattern in my Client-server systems module as well. Relative to my game for this assignment I decided the model would determine logic such as where the player’s current position is and if it overlaps the checkpoint marks. The controller would handle the input such as pressing the arrow keys to move, and the view displays the information. The code implementation for MVC is declaring an instance of each Class and letting the instances handle the rest whilst the gaming is playing.

##### MVC – UML

### Evaluation

I think my game has been pretty successful and I am pretty happy with the result, but I would like to make some improvements. Here are some improvements :

Another improvement that I would like to make is the shooting of the bullets. My bullet speed is calculated using the gradient of the mouse positioning. Therefore, when I shoot vertically in comparison to diagonally the bullet moves at a much faster rate. My math was not strong enough to work this out, but I am currently developing these skills by reading Computer Graphics – Mathematical first steps by P. A. Egerton and W. S. Hall and I hope to improve this in the future.

An improvement I would like to make is the player icon is not restricted to leave the grey path and can pass through the greenery. I have coded some logic to handle this and to stop the user cheating however I believe it would be a lot more of an authentic experience if I ruled the possibility out totally.



Relative to design patterns, an improvement that I would make is using behavioral patterns as well. I have used plenty of creational and a structural pattern, but I believe it would make my assignment a lot more well-rounded if I used a behavioral pattern.

Overall, I thoroughly enjoyed developing a game in JavaFX. As I am a practical learner I enjoyed including the patterns into a project myself and experienced the benefits of the patterns firsthand. This assignment has helped strengthen my coding practices and helped identify areas of weakness such as math. This module also has planted seeds of interest for the 3D Games module in 3rd year.

# References

ORDEUS, E.

**Game programming patterns in Unity with C# | Habrador**

(nordeus, 2021)

nordeus, E., 2021. *Game programming patterns in Unity with C# | Habrador*. [online] Habrador.com.

Available at: <https://www.habrador.com/tutorials/programming-patterns/> [Accessed 1 April 2021].