SWINBURNE UNIVERSITY OF TECHNOLOGY

COS20007

11.2D: Interview Preperation

Lab: LA1-11, Friday, 10:30 AM, ATC621

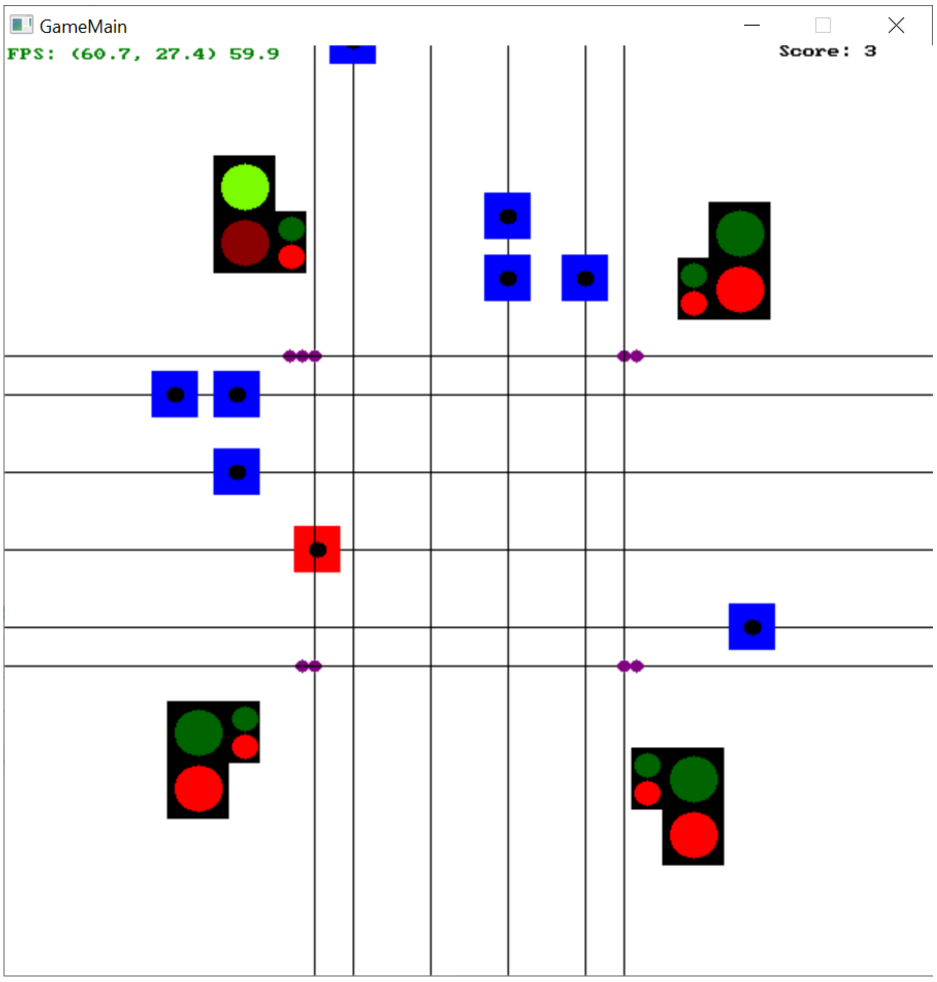
Tutor: Agus Hartoyo

102573957 Matthew Coulter

Custom Program

What is my Custom Program?

For my custom program, I wanted to build something that would be unique and in a genre that I am interested in. I was interested to see if I could build a traffic intersection simulator that could be used by engineers to analyse traffic data to help reduce the amount of traffic delays in the real world. The idea is that the player would be able to control the traffic lights between red and green to let cars and pedestrians through. If cars or people have to wait too long, they would run through a red light therefore the player has to maintain a steady degree of balance. However, I soon noticed that there are so many variables involved in simulating basic traffic behavior. So I have done by best attempt to create a very basic version of it. Because that is the case, it is important that I employed all of the OOP principles as they would allow for greater extensibility, making the original idea more achievable in the long term.



How I have met standards of OOP?

How Have I implemented Polymorphism, Abstraction, Inheritance and Encapsulation?

I was able to borrow the shape classes from the shape drawing application then add extra methods to suit the needs of my software such as IntersectRectangle() and IntersectCircle(). Then, each object contains a container which has a list of Shape. This way, I can make a custom compilation of shapes to form the appearance of a traffic light, car or pedestrian.

I have taken inspiration from the SwinAdventure task where they had GameObject and IdentifiableObject. Instead, I made GameObject and MovingObject which are the parent classes of many class within this game. This way, I can effectively reuse code for a more efficient solution. In iteration 2, I decided the make a car and a pedestrian which are both child classes of MovingObject. I thought this would be a good idea as it shows a deeper implementation of abstraction. Both object behave very similarly, but I wanted to have little differences to better simulate the differences between cars and pedestrians in the real world. For example, cars can crash into pedestrians and other cars but pedestrians can't crash into anything.

Did I run unit tests for my software?

Yes, I found unit tests to be very useful because there is not really another way when the vehicles are constantly moving. Using unit tests, I could manually position one car top be intersecting another car then ruyn the testCollision() function.

How I have gone above the standard

Object Pool Design Pattern

Something that isn't taught in this unit but took my interest is the concept of object pooling. Object pooling is all about controlling the memory and CPU usage by recycling and reusing objects rather than creating new objects. After my first iteration, Agus brought it to my attention that this is a better way to implement the car spawning. Essentially, rather than creating a new object everytime a car spawns that will continue to move indefinitely, the resource usage can be stabilized through the recycling of objects.

This works through the use of a queue and a list for each path. When the game initializes, each path queue is filled up to a capacity of 10. Then, when the object are spawning onto the path (which is controlled by the GameTimer) rather than creating a new object, the object is simply moved from the queue into the list. Then, when the leave the intersection, they are sent back into the end of the queue.

High Distinction Task

I was fascinated by this idea of Object Pooling as I previously mentioned so I saw it as a good opportunity to demonstrate my understanding of it through an educational video collaborating with my brother. He has never done Object Orientated Programming before so I thought that if I were to teach him, he would have some good questions to ask which would allow me to explain my knowledge about the programming concepts.