COS20007

6.3D Custom Program

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

Tutor: Agus Hartoyo

102573957 Matthew Coulter

Main Features

- Player can control the traffic lights to let cars through the intersection
- Spawn vehicles at random times to random paths
- Have the spawn rate become faster and faster
- Cars become angry if they wait too long
- Angry vehicles drive through red lights
- Increase score for each vehicle which gets through the intersection
- Decrease score for angry cars which go through the intersection
- Higher score contribution for cars which don't wait as long
- Cars will crash if their sprites intersect

How it works

In this section, I will explain how each class works, and how gamemain is able to structure the game's operation through their implementation.

GameObject is inherited by any object that is drawn to the screen. It contains a container which is a list of shapes used to construct the shape. This also contains a draw function which draws all of the shapes within the container to the screen. This is all possible through the concepts of inheritance, abstraction and polymorphism.

TrafficLight is constructed of two circles and a rectangle. It inherits Gameobject, overriding it's MouseIn function to determine when the user's mouse is hovering over the trafficlight. This way, in gamemain, when the user clicks on the traffic light, it calls the updatestate method. This updates the colors, and changes the its state, affecting the conditions for MovingObject's Move() method. TrafficLight contains a reference to the list of traffic lights created in gamemain. Likewise, path has a reference to the trafficlight it is assigned to and MovingObject has a reference to the path that is assigned to. This is important as I didn't want to be able to create an object without it existing inside of what it is assigned to. For example, a car cannot be created unless it has a path to spawn onto.

Path has a starting point and an ending point for which the car drives on. For this version, there are only straight lines, no curved lines. This means a static direction can be calculated and assigned, based upon these 2 coordinates, rather than a variable direction for each point on the curve. The calculation method exists inside of ExtraFunctions. The path stores all of the movingobjects that are created on it into a list called movingobjects. This makes it easy to draw all of the movingobjects onto the screen by using a foreach loop inside of a foreach loop in gamemain.

MovingObject contains many functions which determine the behavior of the many features it has. The Move function uses trigonometry and the direction calculated from the path in radians to incrementally step the position pixel by pixel. This is very flexible, meaning that any direction of the path will ensure that movingobject stays on it. The move function contains an if statement, whereby it will only move when the movingobject is not at a red light, if the light is green, if there is no car within 10px of the car in front or if it at a red light and is angry. Appropriate functions are implemented within the movingobject class for this to work accordingly. Move also has a for loop surrounding this if statement for the speed. If the speed is 2, it will step twice, likewise for 4 etc. This is useful as it ensures no pixels are ever skipped, and when the vehicle should stop, such as after a crash, the speed can simply be set to 0.

Getting the movingobject to stop behind another movingobject was difficult. It required creating a function that returns the movingobject in front of the vehicle, given it existed. This is why we must store a reference to the path inside of the movingobject. In combination with the get index function, CarInFront() is able to identify when the vehicle is in front, causing the movingobject to stop.

The crashing feature was also very complicated for multiple reasons; calculation and memory balancing. TestCrash() needs to be run every single frame. However, passing in an array that contains every single movingobject causes drastic memory issues. Hence, using logic by asking "which vehicles can a specific vehicle actually crash into?" we can adjust what gets stored into the movingobjects list. My solution was using a Visible() boolean function. Vehicle's can only crash if they're visible, hence when a vehicle gets through the intersection and out of the frame, it is no longer visible, hence it is no longer taking up unnecessary memory and this feature's memory usage becomes balanced.

By using the ExtraFunctions class, one of the methods: GenerateRandomNumberBetween is used to create the number of seconds of waiting time until the object becomes angry. The object can only become angry if it hasn't passed the intersection, hence the PointCrossed() boolean function. The random time generated is between 5 and 30 seconds, once it hits 0 (if it does), the speed will double, and the object will become red. When this object passes the intersection, it will then take 50 points off the score. If the object isn't angry, the remaining time until becoming angry is what is added to the score.

Shape is an abstract type, inherited by the Rectangle, Circle and Line (similar to the drawing program). This implementation of polymorphism allows each shape to be treated as a shape type, so when it comes time to using the draw function, all of the different shapes in the container can be drawn collectively.

Rectangle implements an intersect feature. Whilst SwinGame has built in methods for intersecting, I had difficulties with that implementation due to type differences. So, I made my own one, which is entirely based off x, y, width and height values. Given the 4 different conditions, this intersect method is called from the TestCrash function in gameobject every frame.

GameMain is where all of the TrafficLights, Paths and MovingObjects are created. They are all drawn every frame. For the spawning of vehicles, I tried for hours trying to implement timers, but nothing seemed to work, so I utilized the frame loop for the timer. The spawn vehicle timer has two variables, the actual time left until next spawn, and the time it gets reset to after spawning. This is to allow the spawning rate to increase. Once the vehicle is spawned, the 'reset to' value is multiplied by 0.92, and spawn timer is then set to that value. Continuing to tick down, it will get faster and faster but never reach 0 because that is where it asymptotes.

Video Link

https://www.youtube.com/watch?v=45_VXxS1y_w