COS20007

6.4HD 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 and pedestrians through the intersection
- Increasing spawn rate that spawn either a pedestrian or a car at a random path
- Cars and pedestrians become angry if they wait too long
- Angry pedestrians and cars drive through red lights
- Score goes up by 1 for car or pedestrian that gets through intersection safely
- Score goes down by 1 for car or pedestrian that goes through a red light
- Cars can crash with cars and people
- A object design pool is implemented for each path

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 sprite which is a list of shapes used to construct the appearance of the object. This references the sprite class which contains a list of shapes alongside draw add and remove methods. The gameobject 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 is the parent class to two child classes: PedTrafficLight and CarTrafficLight. This was for the version 2 of this game. The PedTrafficLight contains paths that spawn pedestrians and the CarTrafficLight contains path which spawn cars. 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 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. Apart of version two, I have implemented object pooling for the path. Originally, there were objects spawning on the path and driving on forever once they left the screen. However, this is inefficient as more resources are used to create the objects. Instead, we can have a queue and a list. The queue is initialized with a size of 10 movingobject when the game runs. When the spawn method is called, this simply moves the object into the list. Once the object leaves the screen, it is sent back to the queue for reuse. This recycling concept saves memory an d whilst it is less significant for computer games, it has a greater impact over low tier processors for phone games.

MovingObject is an abstract class inherited by movingobjects such as pedestrian and car. These child classes implement many methods which control the behavior of the vehicle. 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. Originally, only the visible objects could intersect with each other but this required calculating and clearing a list every single frame which is every inefficient. Now that we have implemented the object pooling, we can have one read-only list that is testing for the intersection every frame. This saves memory as no time is spent on the visible function.

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

This is the video from version 1 which does not show the object pooling and pedestrian inheritance.. The additional features from version 2 will be demonstrated in the interview.