**Assignment 3 Final Report**

The final implementation of the games which stand out are:

The hovercar has its own independent variables which movement is calculated such as speed. It also has a boost variable which acts as turbo for the car. I also have implemented basic health bar which goes down when the car collides with an object with the certain max speed limit.

There is a Non-Player-Character that follows a course throughout the map from start to finish line. I have also included a collision between the NPC and The Player.

The game consists of Stages (created with enum States) which change when the player passes through Checkpoints. The player must pass through each checkpoint in a set order to be able to change the status of the game.

There is User Interface that shows the following information (using stringstreams):

The state of the game, current player speed, boost status, boost warning and countdown to start at the beginning of the game.

A chase camera which allows the player to move throughout the level. I also have included key controls that allow the player to change the camera angle to first and third person.

A basic collision method with all the stationary objects in the game i.e. (checkpoints, walls, water tanks etc.).

When the car collides with an object the

The main technical aspect of the game:

1. The main implementation of the game that stands out is the calculation of the movement which consists of three important factors:

To design the structure of the movement first I created a 2D vector structure which stores all the factors needed for calculating the movement:

hoverCar::Vector2D momentum{ 0.0f,0.0f };

hoverCar::Vector2D thrust{ 0.0f,0.0f };

hoverCar::Vector2D drag{ 0.0f,0.0f };

Then we need matrix X and Y coordinates which will be used the move as main variables to move the car by.

* Thrust:

To calculate the thrust I used a scalar with thrust factor multiplied by the speed variable

and facing vector:

Facing Vector: Vector2D facingVector = { matrix[2][0],matrix[2][2] };

Thrust calculation: thrust = Scalar(carThrustFactor\*speedMeter, facingVector);

* Drag

The drag uses a scalar function Coefficient multiplied by the current momentum which makes it a negative value to act as gravity: drag = Scalar(carDragCoefficient, momentum);

* Momentum

To calculate the new momentum we do a vector calculation by passing the following variables to a function which adds all the X and Z coordinates together and returns them to:

momentum = vectorCalculations(momentum, thrust, drag)

1. The collision methods work as follows:

* 2D Sphere to Sphere and point to Sphere:

To do this calculation you need to take both **objects X coordinate**, **objects Z coordinates and objects radii.** After doing that you calculate the distance between the two objects for **both X and Z** i.e. distanceX = objectXPositionTwo – objectXPositionOne. Then the distance radius by distance = sqrt(distanceX \* distanceX + distanceZ \* distanceZ)

**Then you check them against each other and resolve the collision**:

distance < (objectRadiusOne + objectRadiusTwo)

The only difference between the two collisions is point collision is used to check if the object has passed through it.

* 2D Box to Sphere collision and point to box collision:
* To do this calculation you will need the following coordinates:

sphereXPosition, sphereZPosition, sphereRadius oldSphereXPosition, oldSphereZPosition, boxObjectXPosition, boxObjectZPosition, boxObjectWidth, boxObjectDepth

Where they are used to calculate boundaries for both X and Z to check them against which side is hit e.g.

minimumXBoundary = boxObjectXPosition - (boxObjectWidth / 2) - sphereRadius;

maximumXBoundary = boxObjectXPosition + (boxObjectWidth / 2) + sphereRadius;

if (oldSphereXPosition < minimumXBoundary)result = Collision::leftSide

I expect to get a grade between 60% and 70% as I have complement all the points up to 70% and some of the 70% mark. I expect to score some extra points from including classes, splitting the development into headers and .cpp files which make the construction of the game easier to follow and improve.

