CARZ

Leone Migone, Rendcomb college

Leone Migone

CONTENTS

1 ANALYSIS

1.1 PROBLEM DESCRIPTION

I decided to choose this path for my project because I'm passionate about game development and programming and for my game, I decided to develop a 2D top down racing game, this because I always enjoyed to play racing games through my childhood and I have a lot of good memories related to this type of games, so it rapidly came to my mind in the moment of my decision.

1.2 CLIENT INTRODUCTION

My clients for this project will be Enrico Everitt and Riccardo Manni, even **if** they are part of d**if**ferent demographics they both share a huge passion for gaming and they can give useful feedbacks due to their d**if**ferent experiences in this sector as consumers , Henry is a A-level student that experienced the latest gaming trends and played the most successful games of the second decade of the 21st century, instead Riccardo Manni is an Italian Highschool student definable as a casual gamer, but even **if** he's not as passionate to gaming he'll be useful thanks to his big passion for motorsports and races

1.2.1 First Client Interview – Enrico Everitt

1) "How would you describe your relations with videogames?"
   Since I was a kid I had a big passion for videogames, I usually play on my laptop, my phone but mainly on my Xbox, I play from an average of an hour a day during the week to around 3 during the weekends.
2) "Are you passionate about motorsports?"
   Motorsports are not really my passion, I follow a lot of sports such as rugby and football but I rarely watch motorsports.
3) "Have you ever played racing games and **if** so, which was the last one you played?"
   Yes, I did play racing games, my favourite one that is also the last one I played is Mario Kart 7, I really enjoyed playing game because is designed for casual fun rather than a precise simulation of racing. The inclusion of items like banana peels and shells makes it more entertaining for people like me that are not interested in motorsports.
4) "Which features are necessary in a good racing game?"
   In my opinion, a good racing game in my opinion needs to contain some non-realistic features such as items or turbo to make it more exciting to play otherwise they could bore the user.

1.2.2 Second Client Interview – Riccardo Manni

1) "How would you describe your relations with videogames?"
   Oh, I play video games every now and **then** when I have some free time, I sometimes play F**IF**A with my mates when we are together or a casual mobile game during my break, it's more about having a good time than being super competitive or hardcore about it.
2) "Are you passionate about motorsports?"
   Yes I'm a big fan of motorsports, I never miss a Formula one GP! I have always been fascinated by cars and their components in fact every summer I work part time as a car mechanic in my town and my dream is to be a team manager in a Formula 1 team.

3) “Have you ever played racing games and **if** so, which one is your favourite?”
   Yes of course, when I rarely play video games it’s mostly racing game, I played a lot of them at my friends house and I really enjoy how the models of the car are accurate, my favourite game and the one I play the most is Kart On, I love is simple 2D arcade look and the featuring of a lot of d**iff**erent and variegate tracks in this game
4) “Which features are necessary for you to enjoy a racing game?”
   In my opinion a good racing game needs to feature a local multiplayer since playing with friends make every game more enjoyable, a lot of tracks and maybe a track editor to make the game experience longer.

1.2.3 Clients summary

Both of my clients enjoy playing video games even **if** at d**iff**erent levels and like racing games, from this interviews I could tract important insight that changed appearance and features of my game, such as:

- The game being developed in a bidimensional arcade look.
- The presence of a track editor to make the game experience wider.
- The presence of non strictly realistic features in the game to make it more appealing to consumers that are not necessarily passionate about motorsports.

1.3 RESEARCH

1.3.1 MicroMachines

MicroMachines, released for the first time in **1991** on the **NES**, is a top-down **racing** game with miniature vehicles. The racetracks are unconventionally themed. For example, some races take place on a billiard table while others occur in a garden to follow the theme of a race between proper micromachine and not normal cars, the feature that I took from Micromachine to implement them in this game are:

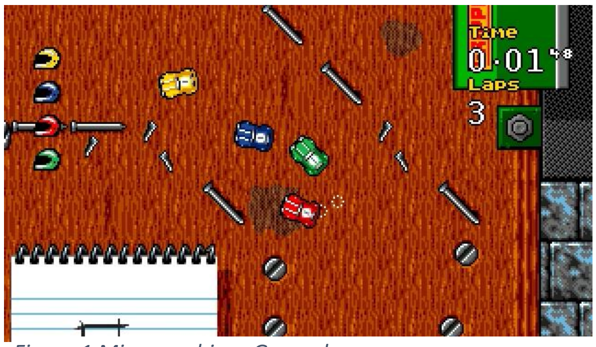


Figure 1 Micromachines Gameplay

- The camera movement, during a race in micromachine the camera follows and zooms on the car that the player is controlling rather than showing the entire race track.

- The HUD(head-up display), in Micromachines in the top right corner of the screen(as shown in Figure 1) we can find the time elapsed since the start of this lap and the numbers of laps remained to complete the race, these data can be useful for the player even **if** it can sacri**fi**ce a more clean and minimalistic look on the screen.
- The presence of a 2 player split-screen mode to play against a friend on the same computer.

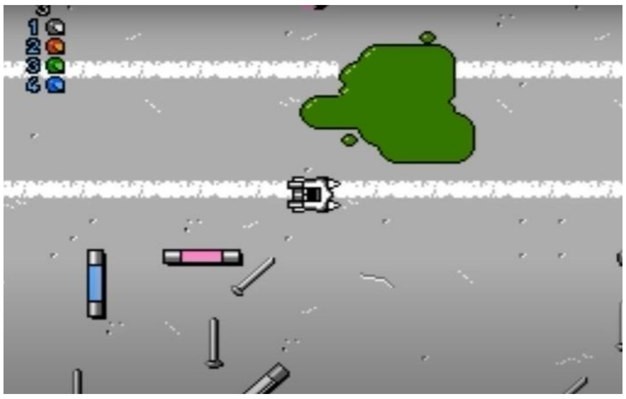


Figure 2 Micromachine's track obstacles

Elements that instead I decided to ignore and not implement in CARZ are:

- The di**ff**erent types of vehicles in the game, In MicroMachines we can count 9 di**ff**erent types of vehicles including for example powerboars and tanks that have their own shooting system as well, CARZ instead will focus on more classical automobiles but still keeping a di**ff**erence between them regarding their attributes such as speed and braking.
- The presence of obstacles in the race track, MicroMachine race track are always filled with di**ff**erent obstacles such as oil stains(as shown in Figure 2)to slow down the players but in my perspective they risk to make the game too di**ffi**cult and so non enjoyable to a wide range of players with di**ff**erent skill sets.

1.3.2 Mario Kart 7

Mario kart 7, released for the first time in 2011, is a racing game developed by Nintendo, that however, distances itself from a realistic driving simulator thanks to its cartoon look and the implementation of imaginative elements such as powerups. The elements that I took reference from to implement in CARZ are:

Figure 3 Mario Kart 7 gameplay

- The dri**f**t and boost system, in Mario kart by dri**f**ting you accumulate energy to use in a boost that increases your car acceleration for a certain amount of time that depends on the quantity of energy that you accumulated
- The possibility of pick your car between others with other graphics and aspect

Some traits of this game that instead I didn't decide to pick up are:

- The camera position, in Mario Kart 7 a third person camera its used instead in my game the camera will shoot top down to give a clearer view of the player and opponents cars ant their position in relation to the racetrack.

1.3.3 Kart On

Kart On, featured on various websites it's a web browser game that uses Adobe Flash Player, it's a top down kart racing game featuring various modes and numerous tracks, the element that I decided to take inspiration from in this game is:

- The gameplay based on simple car movements and not to realistic collisions to prevent the game to be less enjoyable by casual consumers due to being too simulative rater than arcade

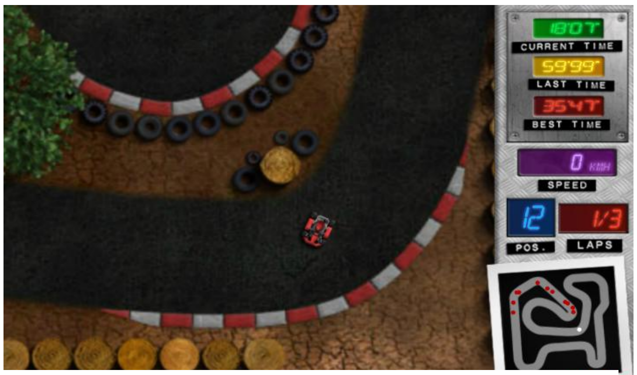


Figure 4 Kart On gameplay

1.3.4 Games Research Summary

The main feature that I took from these games that inspired me are:

1. The camera positioning and movement from MicroMachines.
2. The drift and boost system from Mario Kart 7 .
3. The presence of different cars and the ability to pick them in the menu of Mario Kart 7.
4. The arcade movement and collision system of Kart On.
5. The 2 player split screen mode from MicroMachines.

1.3.5 Track editor

my project will feature maps already preloaded into the game but also the possibility of creating your own thanks to the presence of a track editor, this editor will need to be based on a system to allow the users to shape the track similar to what they had in mind. I had to consider different options to fulfil my aim of having a track editor that is easy and intuitive to utilize but that at the same time gives you an adequate number of different options for the look of your map.

1.3.5.1 GRID METHOD

The first method that I took in consideration and I decided to pick at the end is what we could define as the "GRID METHOD", it consist in a 9x6 grid of square where each of them consists in a scaled down part of the actual map that we are going to use in the game, for every square we have 6 different tiles that comprehend 2 version of a straight road, one orientated in horizontal and one upwards, and 4 versions of a curve rotated of 0°,90°,180° and 270° degrees as shown in the Figure 5, in conclusion when the tiled are placed the user will press enter to fill the gaps in the grid with a plain tile and the project will check if the map created it's valid and so passable from start to finish without touching gaps or going out of the road.

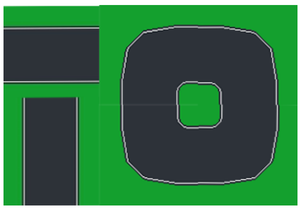


Figure 5 Tiles

1.3.5.2 WAYPOINTS AND BEZIER CURVES

Another method that I looked at before picking the grid method is to a create a map by using way points but I ended up discarding these option because it would've been less intuitive and harder to make the tracks for the users even if they could've been more variated, with this method the user would have been able to select various point the space that the game would connect with a direct line to make a closed circuit at this point the user would be able to change

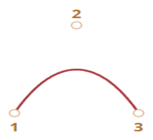


Figure 6 representation of a Bezier curve the inclination of the lines and transform them in Bezier curves(as show in Figure 6) by using a third point.

1.3.6 Track storing method

To store the tracks in the game and to prevent them from being erased when the game is closed, I opted to use a binary file, the tracks created by the user using the grid method are serialized as an object and written in a binary file contained In the game repository.

1.3.7 IDE and Framework - Microsoft Visual Studio 2022

To develop my project, I decided to use Visual Studio 2022 as my integrated development environment (IDE) since it's support a variety of languages such as C# in which my project has been written, For my framework I opted for a cross platform open source development library called Raylib.

1.3.8 Pathfinding Algorithms

Items will be an integral part of the gameplay of CARZ to distinguish it from a simple race game and give and to make it more competitive and exciting to play, some of these items will comprehend the capability of aiming at the closest enemy such as an homing missile, to add this feature to the game I took in consideration some of the algorithms that I could use for those items to find and follow the closest enemy:

1.3.8.1 DJIKSTRA ALGORITHM

Having a starting and a target node the Djikstra algorithm finds the shortest path between them in a graph, we visit the starting node and we update the distance from it of his neighbors, **then** we continue by visiting the unvisited neighbor with the lowest distance from it and so on by continuing this process until all nodes are visited, at that point we'll know the shortest path that takes from the starting node to reach each of the nodes.

1.3.8.2 A SEARCH ALGORITHM*

The A* Search is commonly used to find the shortest path in a grid, it does it by using 3 values about each box in the grid, the G cost that consist in the distance of that node from the starting one, the Heuristic cost that consist in the distance from that point to the target node and F cost that represents the sum of the G and the Heuristic cost. By knowing this values the algorithm will traverse the grid by going always to the cell of the grid with the lower F cost until it reaches the target node and it can so define the shortest path. Figure 8 shows a graphic example of a A* Search applied to a grid, where in every cell the number on the top left is the G cost, the one on the top right is the Heuristic cost and the one in the middle is the represent the F cost.

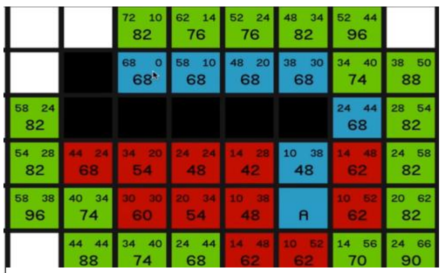


*Figure 7 A*Algorithm in a grid*

1.4 Modelling

1.4.1 MAIN MENU

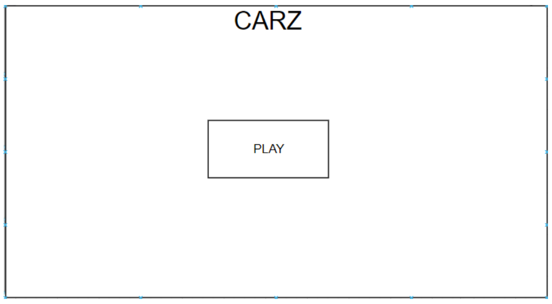


Figure 8 Main Menu (first version)

The first screen that will be shown to the user is going to be the main one as illustrated in Figure 8, under the title “CARZ” we find the button “PLAY” that make us go to the page illustrated in Figure 10 that consist in the track selection.

Client Comments:

- The screen is very plain. Will there be any animations to show what the game looks like?

To give a more interactive and eye catching look at this menu in the background of this menu there will be shown an animation that adds graphic details to the menu and gives a peek of the game main theme. This animation will consist of a starfield generation algorithm where the stars of random colors will change the velocity in which they leave the screen in variation of the position of the mouse on the screen.

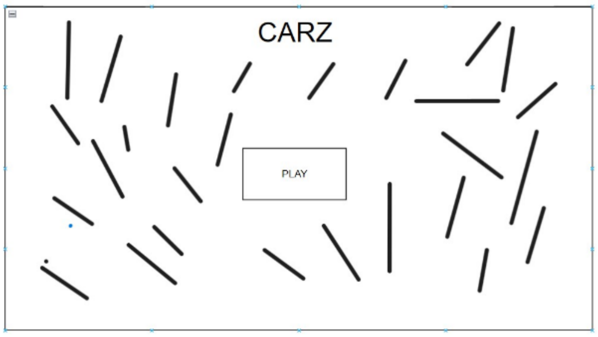
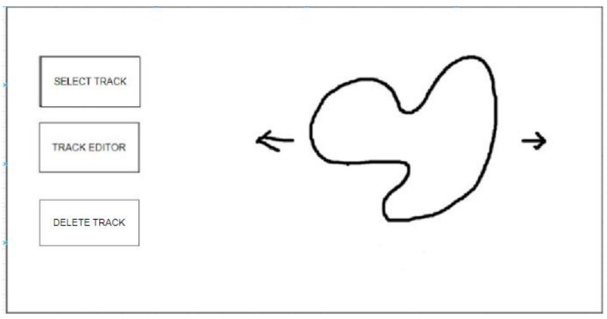


Figure 9 Main menu (definitive sketch)

1.4.2 TRACK SELECTION



In this menu (as shown in Figure 10) we can select one of the tracks already present in the game by scrolling through them with the two arrows, while scrolling through them we’ll be able to see above the map sketch the name of the track **if** added when created, when we have decided which map we want to use we can either press “SELECT TRACK” that brings us to Figure 12, press “DELETE TRACK” that will delete the track from the binary file where its saved and so not be accessible anymore or **if** we are not satisfied with the maps already present in the game we can build our new one that will be added in the scrolling menu, **if** we want to do so we can do it by pressing the button that says “TRACK EDITOR” that brings us to the screen in Figure 11.

Client Comments:

- I like the way of choosing the track. It looks easy to scroll through.

1.4.3 TRACK EDITOR

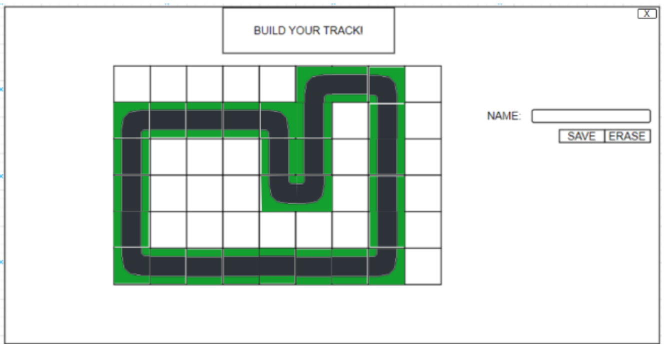


Figure 11 Track Editor

In this screen (Figure 11) we can fill the map grid to create our own track by using two di**ff**erent "pieces" a curve and a straight road, to help us make our track as we like it we can rotate all the pieces in our preferred direction to do so we use three di**ff**erent buttons on our mouse, left click to place a curve tile in the cell of the grid correspondent to the mouse position or **if** its already present in the cell it will just rotate it of 90°, right click to instead place a straight road tile and rotate it case its already in the cell pressed and the middle button to remove the tile in the cell correspondent to the mouse position when the button is pressed. !When we are done with the grid we can name our track and save it by pressing the button "SAVE", to be saved correctly it will need to be validated by an algorithm that checks that is a passable and closed track, so **if** it pass the validation it will be shown with the other race tracks in the TRACK SELECTION and we will be brought back to the screen in Figure 9 otherwise **if** it's not an eligible map it will just be erased and we'll stay in the same screen. **If** we are not satisfied with our track we can either press the button "ERASE" that will empty the cells of the grid so we can restart to modi**f**y our map or we can press the button "X" to go directly back to the TRACK SELECTION screen illustrated in Figure 10 without saving any new map

Client comments

- What will happen **if** the track doesn't link up. Can you still save it?

  *No it won't be possible to save tracks that don't link up, when "SAVE" is pressed the game validates the track checking the neighbours of the pieces to check **if** each them has two valid neighbours and after we also traverse the track road to check that there is only one closed linked track since the first process wouldn't detect has an error the presence of two closed linked tracks. **If** the validation process goes wrong an error message will be shown and the track grid cell will be set as empty again.*

1.4.4 CAR SELECTION

In this screen we will choose between one of the cars to use it in the race by scrolling through them using the arrow buttons, player one will chose using the arrows on the left side of the screen and the second player will choose using the one on the right, when we decide which cars we want to use we can press the button "SELECT CAR" this will bring us to the screen illustrated FIGURE 12 but we also have the option to go back to the TRACK SELECTION (Figure 10) by pressing the "X" button in the top right corner of the screen.

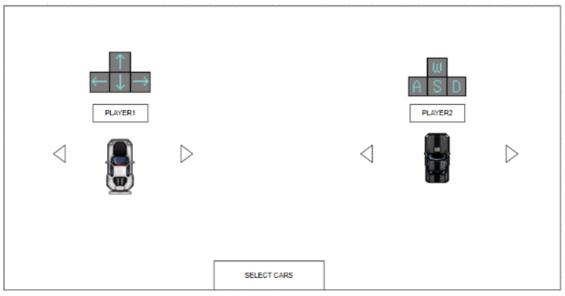


Figure 12 Car Selection

1.4.5 GAMEPLAY HUD

When we reach this screen the gameplay and the race will automatically start, Figure 13 illustrates a sketch of the gameplay, the game feature a split screen in which in each we can find the laps completed, the lap time and, best lap of each player (player 1 in the middle of the screen and the second player that can both be human or a bot on the right side) and the quantity of boost available for each car by using a red and yellow rectangle. When one of the cars finishes the lap we go to the winner screen in figure 14.

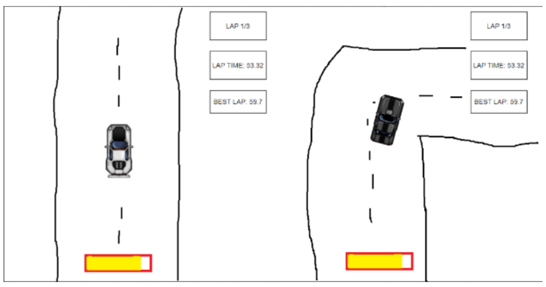


Figure 13 Gameplay HUD

1.4.6 WINNER SCREEN

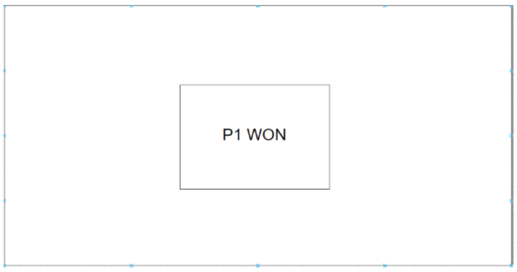
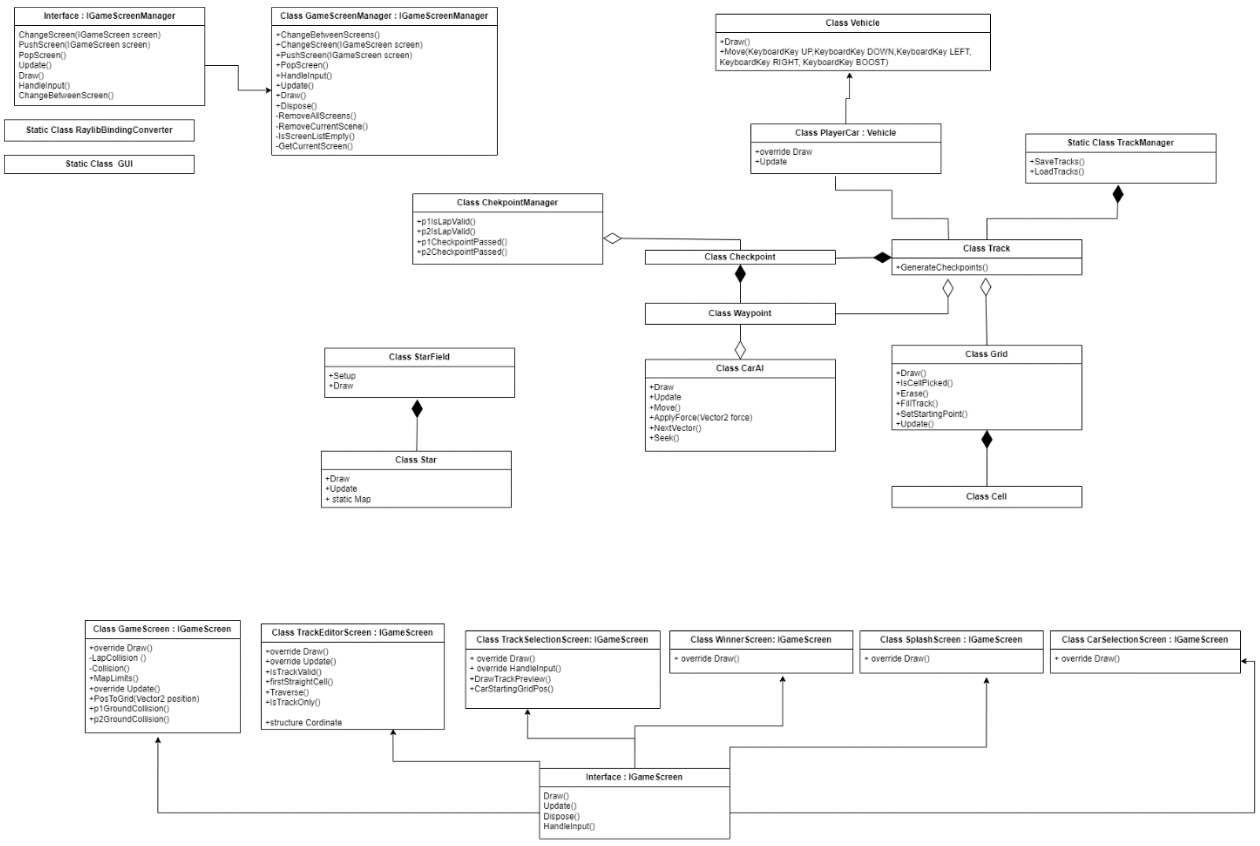


Figure 14 Winner Screen

This screen will show in its middle the winner of the race that has just ended, **if** we press on the button that says who's the winner the game will bring us to the Track selection screen illustrated in Figure 10.

1.5 UML DIAGRAM



1.6 Objectives

1. User will be shown the first screen.
   - 1.1 A starfield generation animation is shown in the background of the splash screen.
     - 1.1.1 Stars will spawn in random point of the screen and move towards the user
     - 1.1.2 The speed of the stars in the animation will increase progressively according to the mouse x position
   - 1.2 The first screen will contain the button “PLAY”.
     - 1.2.1 The button is pressed by left clicking it.
     - 1.2.2 When “PLAY” is pressed, the user gets brought to the Track Selection screen.
     - 1.2.3 The user will exit the game if the ESC key on the keyboard is pressed.
2. User will be shown Track Selection screen.
   - 2.1 This screen will feature 3 buttons (“SELECT TRACK”, ”TRACK EDITOR”,”DELETE TRACK”)and 2 arrows.
   - 2.2 The Menu shows a preview of the look of the current track by reading the content of the binary file where we the tracks are stored
   - 2.3 The buttons and the arrows are pressed by left clicking on it.
     - 2.2.1 When “SELECT TRACK” is pressed the user gets brought to the Car Selection Screen
     - 2.2.2 When “TRACK EDITOR” is pressed the user gets brough to the Track Editor Screen
     - 2.2.3 When “DELETE TRACK” is pressed the current track showed in the menu gets deleted from the list of tracks of the game
     - 2.2.4 When the left or right arrow are pressed, we scroll through the list of different tracks available and so it changes the current track.
     - 2.2.5 every time we press one of the arrow it will update if more that one track is present in the list.
     - 2.2.6 The Menu shows on top of the preview of the current track his name by reading the content of the binary file where the tracks are stored, so every time we press one of the arrows it will update if more than one track is present in the game.
     - 2.2.7 The user will exit the game if the ESC key on the keyboard is pressed.

3. User will be shown the Car Selection screen.
   - 3.1 This screen will feature 4 arrows and 2 buttons (“X”, “SELECT CARS”).
   - 3.2 The screen will show the image of the current car for the first and second player.
   - 3.3 The buttons and the arrows are pressed by left clicking on it.
     - 3.3.1 When one of the two arrows in the left side of the screen are pressed, we change the current car of the first player.
     - 3.3.2 When one of the two arrows in the right side of the screen are pressed, we change the current car of the second player.
     - 3.3.3 When an arrow its pressed the image showing the previous Car will update to the current one.
     - 3.3.4 Relative Commands for player 1 and 2 should be shown respectively above the two cars
     - 3.3.5 When “SELECT CAR” is pressed, the user will be taken to the Gameplay screen.
     - 3.3.6 When “X” is pressed, the user will go back to the Track Selection screen.

4. User will be shown the Track Editor screen.
   - 4.1 This screen will feature an interactive 9x6 cells grid, 3 buttons (“SAVE”, “ERASE” and “X”) and a text box.
     - 4.1.1 The user will be able to use the interactive map using the mouse buttons.

4.1.1.1 When the user left clicks on an empty cell or a straight road cell, a curve texture will be inserted into the cell.
4.1.1.2 When the user right clicks on an empty cell or a curve texture cell, a straight road texture will be inserted into the cell.
4.1.1.3 When the user left clicks on a curve cell, it will rotate the texture by 90°.
4.1.1.4 When the user right clicks on a straight road cell, it will rotate the texture by 90°.
4.1.1.5 When the user middle clicks on any sort of road it gets set as a empty cell
4.1.1.6 When the user press Enter on the Keyboard, the empty cells of the grid will be filled by a grass texture.

4.1.2 When "SAVE" is pressed and the track is valid, the track created in the grid will be saved with the name inserted in the text box in a binary file and the user will be brought to the Track Selection screen.
4.1.3 When "SAVE" is pressed and the track is not valid, a error message will be shown, and if pressed it will disappear leaving the user to the track editor and set every cell of the grid to be an empty cell.
4.1.4 When "ERASE" is pressed, every cell of the grid gets substituted with an empty cell.
4.1.5 When "X" is pressed, the user will go back to the Track Selection screen.

5. User will be shown the Gameplay screen.

5.1 first and second player cars move accordingly to the pressed keys(if the user chose to play against the cpu the second car will move automatically)
5.1.1 if "W" is pressed the first player car accelerates forward.
5.1.2 if "S" is pressed the first player car brakes.
5.1.3 if "A" is pressed the first player car rotates anti clock wise and the first player boost recharges.
5.1.4 if "D" is pressed the second player car rotates clock wise and the first player boost recharges.
5.1.5 if the SHIFT key is used and the first player boost bar is not empty, the first player car uses its boost.
5.1.6 if the UP ARROW key is pressed the second player car accelerates forward.
5.1.7 if The DOWN ARROW key is pressed the second player car brakes.
5.1.8 if the LEFT ARROW key is pressed the second player car rotates anti clock wise and the second player boost recharges.
5.1.9 if the RIGHT ARROW key is pressed the second player car rotates clock wise and the second player boost recharges.
5.1.10 if the right CONTROL key is used and the second player boost bar is not empty, the second car player uses its boost.

5.2 The game shows 2 different screens(split screen).
5.3 The left half of the screen will refer to the first player.
5.3.1 The left half will show on his right top side the laps percurred by the first player.
5.3.2 The left half will show on his right top side the lap time of the first player.
5.3.3 The left half will show on his right top side the best lap time at which the first player has percurred the track in this game.
5.3.4 The left half of the screen will show the camera on the first player car and follow his position.
5.3.5 The left half will show the boost quantity of the first player in the middle low part of this half of the screen.

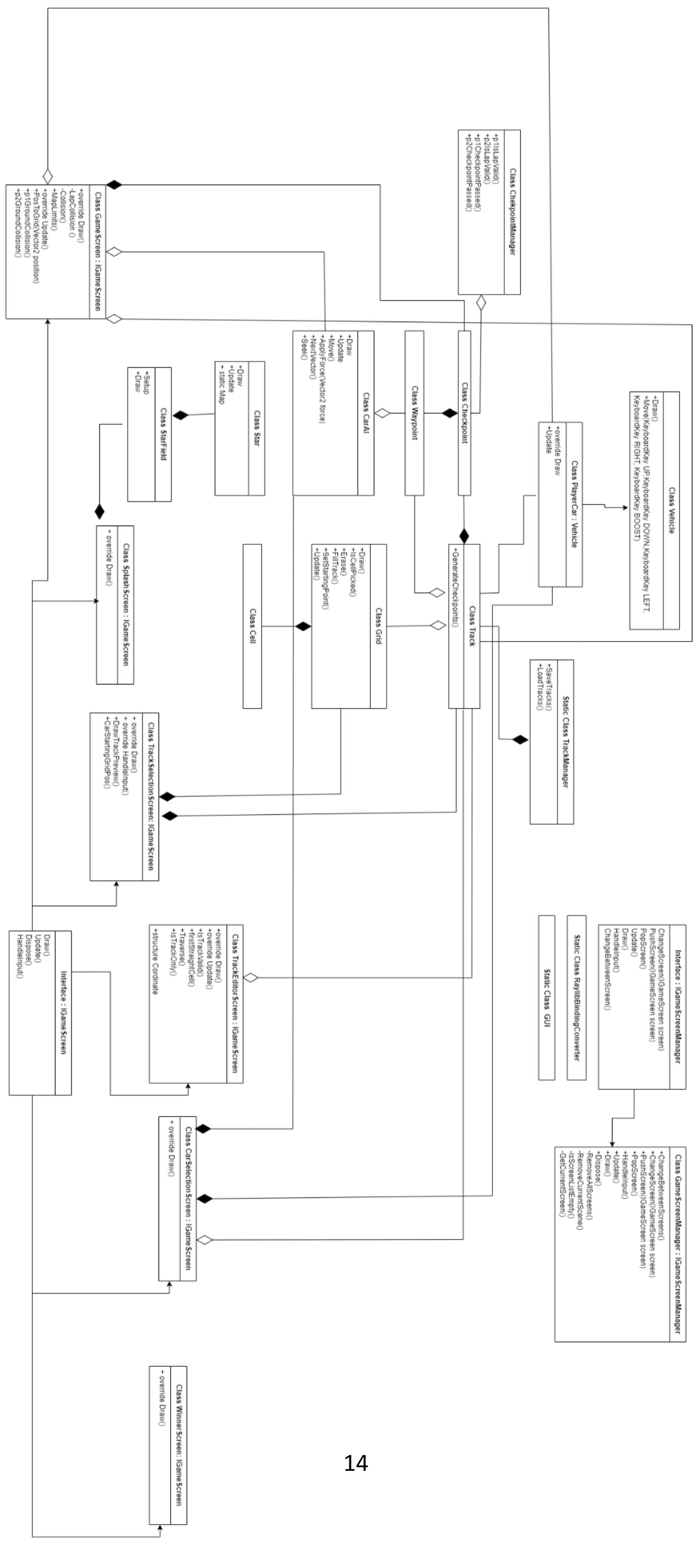
5.4 The right half of the screen will refer to the second player.
5.4.1 The right half will show on right top side the laps percurred by the second player.
5.4.2 The right half will show on his right top side the lap time of the second player.

        5.4.3 The right half will show on his right top side the best lap time at which the second player has percurred the track in this game.
        5.4.4 The right half of the screen will show the camera on the second player car and follow his position.
        5.4.5 The right half will show the boost quantity of the second player in the middle low part of this half of the screen.
    5.5 When one player completes a valid lap the lap time gets reset, the best lap time gets updated if the lap time is better than the previously registered one, and the completed laps number gets updated.
    5.6 When one Player completes all the laps in the race User will be shown the Winner Screen.

6. User will be shown the Winner Screen
    6.1 P1 WON or P2 WON will be shown accordingly on who won the race.
    6.2 If the button with the name of the winner is pressed It will bring User back to the Track Selection Screen.

2 DESIGN

2.1 OOP DIAGRAM AND CLASSES DESCRIPTION



Star	• This class contains 3 methods, Update(), Draw() and Method(). • This class defines the position of each star instantiated in the starfield by assigning them a random value for their x, y and z position within the screen boundaries in his class constructor. • By using the Update() method, it updates the star position the speed of the starfield(Starfield.Speed) from its current z value simulating the moving of the stars towards the viewer. • By using the Draw() method, it draws the star moving on the screen of a randomly generated color.
Starfield	• Displays on the screen a starfield animation that show stars moving on the screen towards the user • This class contains 2 methods, Setup() and Draw(). • This class manages an array of Star objects of a fixed size of a 1000 • it firstly initialize each of the stars composing the array in the Setup() method. • it draws the stars on the screen based on their positions using the draw function present in each Star object. • It updates the stars position based on the speed. • It dynamically adjusts the Speed based on the mouse X position, creating a faster movement when we get closer to the right edge of the screen and so closer to the play button.
Cell	• This class defines the characteristics of a single Cell of the track grid. • It stores the dimension of each cell of 112x112 pixels with the variable width and height. • It indicates the property of each cell, **if** they have been traversed, **if** they are the cell of the track that correspond to the starting/finishing line, the type of road tile that they are(e.g. straight road, curve, or an empty tile) and the rotation of the texture correspondent to their road tile.
Grid	• This class represents the grid structure used for creating the racetrack. • It manages a 2D array of Cell objects and provides functionalities for drawing the grid, handling user input for mod**if**ying the cells using the track editor and managing track-related features such as automatically setting the starting point. • It defines the fixed width and height of the grid that is represented with a 2D array. • It draws the outline of each cell and the corresponding texture of each cell in its given rotation, filling with grass textures empty cells. • It handles the input of the user by assigning the road tile (correspondent to the mouse button pressed) to the cell of the grid where the mouse is positioned
Track	• Manages and contains Track characteristics such as his Grid the track name and List of Waypoints relatives to the map structure. • Generates the checkpoint,in the GenerateCheckpoints() method, utilised during the race to validate each lap and avoid cheating.
Waypoint	• Defines and manages the characteristics of a waypoint creating his x and y variables that correspond to his position.

TrackManager	• This class composed by the two methods SaveTracks() and LoadTracks(), it takes care of saving the passed list of track created by the user in a binary file and when needed accessing the same binary file to load the tracks and return a list composed by them.
Checkpoint	• This class defines a checkpoints and their correspondent hitbox using the Rectangle variable called “checkRect”
CheckpointManager	• This class manages the logic for validating and tracking completed laps for the two players (P1 and P2) in a race using two queues and a list of Checkpoints. • The two queues of Checkpoints “P1Queue” and “P2Queue” store the sequence of checkpoints each player needs to pass through for a valid lap. • The list of Checkpoints “CheckpList” represents all the checkpoints defined for the track • With p1CheckpoinPassed() and p2CheckpointPassed() methods, It dequeues the first checkpoint from P1's or P2’s queue, indicating it has been passed. • With p1IsLapValid() and p2IsValid(), it checks it the lap has been completed and so the relative player checkpoint queue is empty in that case it validates the lap, decrease the amount of remaining p1 or p2 laps and refills the queue with the checkpoints for the next lap using the list
Vehicle	• This Class Defines the characteristics and attributes of the vehicle used in the game such as position, acceleration, braking, maximum speed, the speed at which the vehicles rotates, the boost level and the maximum boost. • It contains the draw method of the vehicle • It defines the movement system of the vehicle in relation to the user input in the Move() method, so that: The LEFT and RIGHT correspondent passed Key rotate the vehicle and recharge the available boost while pressed The UP correspondent passed Key applies acceleration in the forward direction based on the vehicle current rotation The DOWN correspondent passed Key applies braking force and slows down the vehicle. The BOOST correspondent passed Key temporarily boosts acceleration and top speed of the vehicle while pressed. This method also limits speed to the set maximum, applies friction to gradually reduce velocity over time and update vehicle’s position based on the velocity at which its moving.
PlayerCar	• This class inherits from the Vehicle class and represents the player controlled vehicle in the race. It manages the player's input keys, contains a method to draw the car and updates the player's hitbox rectangle based on the vehicle's rotation for Collision detections purposes.

CarAI	• This class represents The cpu controlled car that can be used in game. It manages car's movement, his seeking behaviour towards waypoints and collision detection. • Implement the movement system through the methods Move() and Seek(), and calculating the next waypoint's vector it implements a seeking behaviour. • Adjust the car rectangle position based on car rotation for collision detection. • It contains the draw method of the CPU car.
SplashScreen	• This class defines the splash screen implementing the IGameScreen interface. • It displays the initial screen of the game with a starfield generation animation in the background. • Shows a "PLAY" button to transition to the track selection screen.
TrackSelectionScreen	• This class defines the track selection screen implementing the IGameScreen interface. • Manages Track selection for the game. • Provides using RayGui the UI elements and the input handling to navigate through different tracks,delete them, select one and go to the car selection screen or create one going to the track editor. • Allows players to preview the current track by drawing it in a scaled version. • Calculates the car's starting grid cell for the track selected
TrackEditorScreen	• This class defines the track editor screen implementing the IGameScreen interface. • Provides UI elements that allow the players to enter a track name, save the track in the binary file or erase it, all using RayGui. • Allows players to insert, rotate and remove road tiles in the track grid to compose a track using the mouse buttons. • Traverse the track as a graph when the buttons "SAVE" is pressed to avoid saving a track that is not valid.
CarSelectionScreen	• This class defines the car selection screen implementing the IGameScreen interface. • CarSelectionScreen class manages cars selection. • Provides car selection for Player 1 and Player 2 that can be either controlled by a user or by the CPU. • Provides UI elements that allows players to go trough the different cars textures using RayGui. • Draws the different car textures on screen.
GameScreen	• This class defines the game screen implementing the IGameScreen interface. • GameScreen class manages a the two player split screen race. • Draws the split-screen view with each player's camera, track, car, and lap/time informations. • Checks for completed laps and update lap counts and best lap time for each player. • Handles car to car collisions and friction between the car and different types of ground. • Prevents cars from going off track and out of the map boundaries.

	• Updates players cars position.
WinnerScreen	• This class defines the screen that shows the race winner player implementing the IGameScreen interface. • Shows the name of the winner of the race in the middle of the screen. • Brings you back to the track selection screen **if** the text box with the name of the winner is pressed

2.1.1 Interfaces

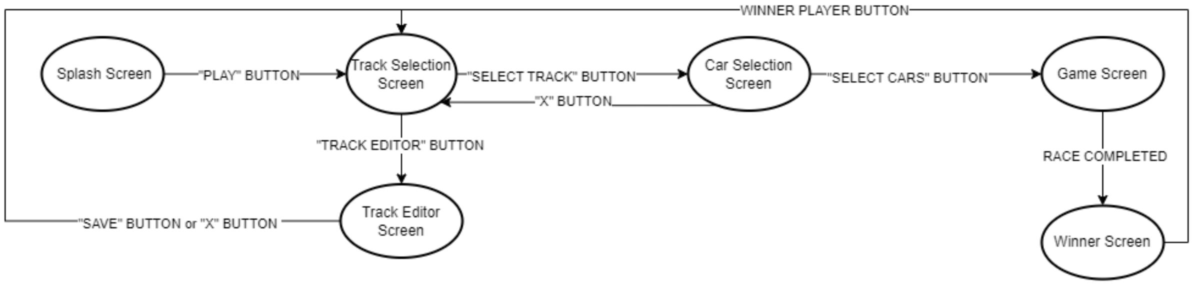
IGameScreen	1.This interface defines the characteristics of a game screen. 2.It contains 4 methods: Draw(),Update(),Dispose() and HandleInput().
IGameScreenManager	1.This interface defines the characteristics of the screen manager of the game. 2.It contains 12 methods: ChangeScreen(), PushScreen() ,ChangeBetweenScreens() ,PopScreen(), HandleInput(), Draw(), Update(), Dispose(), RemoveAllScreen(), RemoveCurrentScreen(), IsScreenListEmpty(), GetCurrentScreen().

2.2 GAME LOOP AND RAYLIB

The game utilizes the Raylib-CsLo framework which is C# wrapper for the Raylib library, Raylib is a lightweight and easy-to-use game development library written in C. It provides a simple and efficient API for handling window creation, input handling, graphics rendering , and other game-related functionalities, the game loop is initialized using the raylib initWindow() function and so setting it to the stored screenwidth and screenheight values, it sets up the game textures and and the first game screen using the screen manager. The main loop runs until the escape key is pressed, in the loop the game is Drawn and Updated.

2.3 SCREEN MANAGER

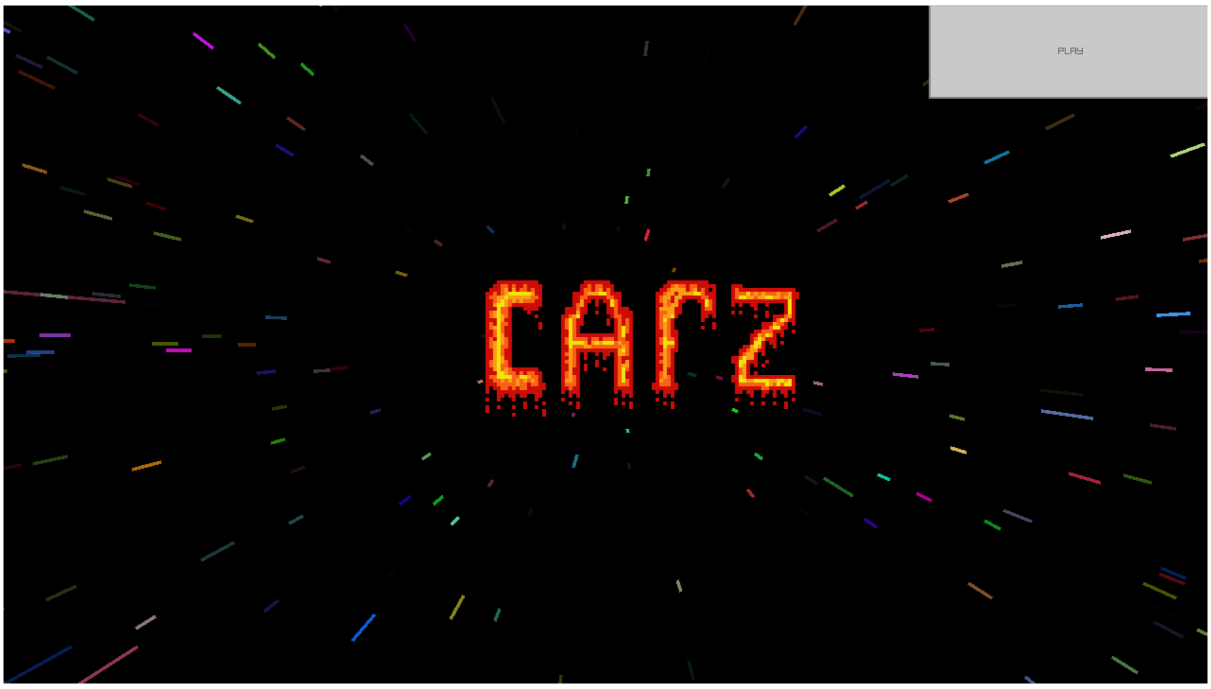
The game screens are managed in the solution by the ScreenInterfaces and the GameScreenManager class. In the ScreenInterfaces class the IGameScreen and the IGameScreenManager interfaces are defined, the IGameScreen interface defines the functionalities that every game screen class(and they inherit) in your application should implement. specifying what methods a class must have (Update, Draw, Dispose, HandleInput) but not how they should be implemented. Allowing for flexibility in creating different types of game screens while ensuring they all provide the necessary functionalities for the game screen manager to interact with them, the IGameScreenManager interface instead defines the functionalities expected from the game screen manager class specifying the methods that this class must have. The GameScreenManager class inherits from the IGameScreen interface and manages a stack-like structure using a List<IGameScreen> to store screens and provides methods for pushing, popping, handling input, updating, and drawing the current active screen, we can find an implementation of it in each game screen class so that it can be called to push the next screen/pop the current one and other actions, when we receive an input in the current screen such as a button being pressed.



2.4 USER INTERFACE AND RELEVANT ALGORITHMS

2.4.1 MAIN MENU/SPLASH SCREEN

The main menu screen has only one button "PLAY" that the user can left click with their mouse that brings user to the track selection screen, to avoid a too plain look as suggested by the client the screen features in the background a starfield generation animation



2.4.1.1 Starfield Generation Algorithm

To implement in the game a starfield generation algorithm and give a less flat look to the splash screen I used two classes: Star and Starfield

2.4.1.2 Star class

```
CLASS Star
 FLOAT x
 FLOAT y
 FLOAT z
 FLOAT previousZ
```

Figure 15 Star Class variables

The 4 variables declared at the start of the Star class are all floats and thet represent respectively, the position of the star in the x, y and z axis when generated and previousZ is the screen depth of the star at the moment in which its spawned and it will used to trace the trail of the star from spawning to going outside of the screen.

The constructor of the Star class assign to the variables x, y and z a random value to give the star a initial position in between the screen boundaries, and sets the value of previousZ to be equal to the just assigned value of z.

```
CONSTRUCTOR Star()
  x ← Raylib.GetRandomValue(-screenwidth,screenwidth)
  y ← Raylib.GetRandomValue(-screenheight,screenheight)
  z ← Raylib.GetRandomValue(0,screenwidth)
  previousZ  ← z
END CONSTRUCTOR
```

```
PROCEDURE Update()
  z ← z – Starfield.Speed
  IF z < 1
    x ← Raylib.GetRandomValue(-Program.screenwidth, Program.screenwidth)
    y ← Raylib.GetRandomValue(-Program.screenheight, Program.screenheight)
    z = screenwidth
    previousZ ← z
  ENDIF
END PROCEDURE
```

In the procedure Update() we decrease the value of z(the star screen depth) by Stardield.Speed, the speed instantiated in the Starfield class, Moving so the star closer to the user. **If** the value of the star is less than 1 and so it goes outside of field of view of the user, the same star will be randomly repositioned in a random location of the screen and at the original depth to make it look further away.

```
FUNCTION FLOAT Map(FLOAT value, FLOAT low1, FLOAT high1, FLOAT low2, high2)
  FLOAT normalizedValue ← (value – low1)/(high1 – low1)
  FLOAT mappedValue ← low2 + (normalizedValue * (high2 – low2))
  RETURN mappedValue
END FUNCTION
```

The Function Map() remaps the value the value passed trough linear interpolation and returns the mapped value.

```
PROCEDURE Draw()
  INT sx ← Map(x/z,0,1,0,screenwidth)
  INT sy ← Map(y/z,0,1,0,screenheight)
  FLOAT size ←  Map(z,0,screenwidth,14,0)

  INT px ← Map(x / previousZ, 0, 1, 0, Program.screenwidth)
  INT py ← Map(y / previousZ, 0, 1, 0, Program.screenheight)
  previousZ ← z
  RANDOM random ← new RANDOM()
  INT r ← random.Next(0,255)
  INT g ← random.Next(0,255)
  INT b ← random.Next(0,255)
  INT a ← random.Next(0,255)

  COLOR color ← new Color(r,g,b,a)
  Raylib.DrawLineEx(new VECTOR”(px + Program.screenwidth / 2, py + Program.screenheight / 2)
  , new VECTOR2((sx + Program.screenwidth / 2, sy + Program.screenheight / 2),3.5f, color)
END PROCEDURE
END CLASS
```

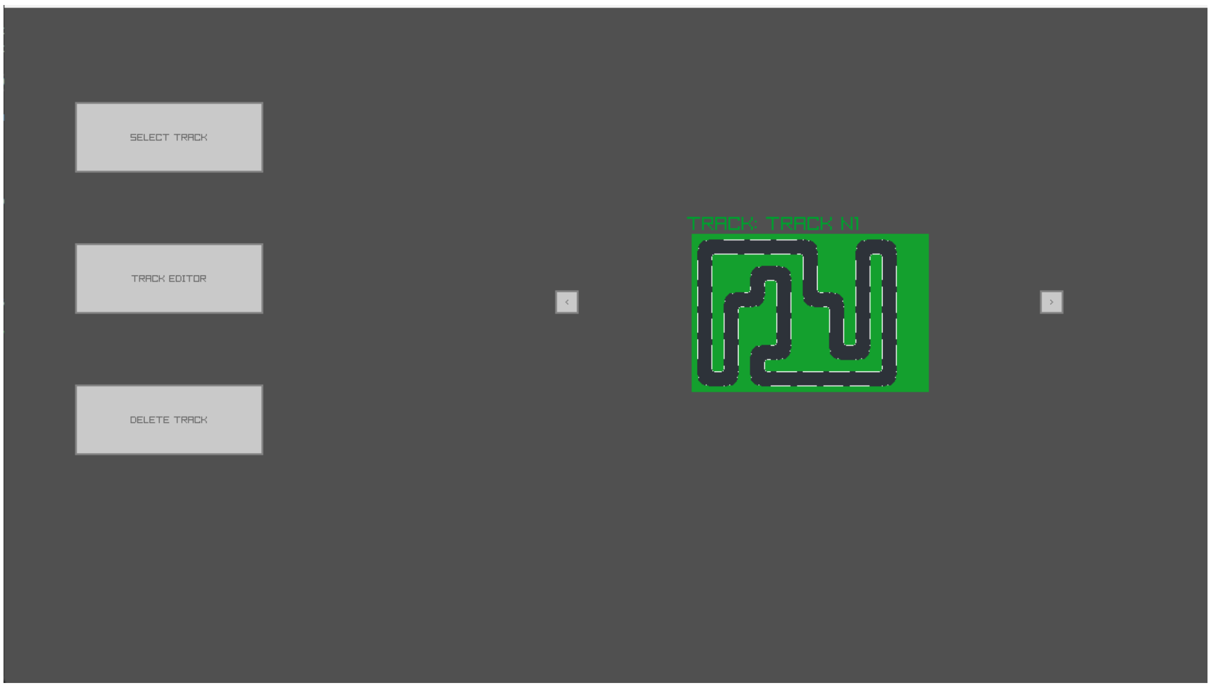
This procedure Draw() firstly calculates the position of the star on the screen (sx,sy) and his size z remapping them using the Map function, draws the trail of the star from his spawning position (previousZ) to the current depth of the star in different randomized colors.

```
CLASS StarField()
  FLOAT Speed
  ARRAY[1000] of STAR stars

  CONSTRUCTOR Star()
    Setup()
  END CONSTRUCTOR

  PROCEDURE Setup()
    FOR i ← 0 to stars.Length
      stars[i] ← new Star()
    ENDFOR
  END PROCEDURE

  PROCEDURE Draw()
     Speed←Star.Map(Raylib.GetMouseX(),0,Program.screenwidth,0,55)
     FOR i ← 0 to stars.Lenght
       stars[i].Update()
       stars[i].Draw()
     ENDFOR
  END PROCEDURE

END CLASS
```

The starfield class manage the field of stars of the animation, defines the speed to control the stars movement and an array to hold the star objects. In his constructor we call the Setup method that is used to instantiate all the stars in the array through a for loop. Lastly the draw method is used to vary the speed of the stars based on the mouse position of the user and in the splash screen to draw and update all the stars in the array showing so the starfield animation.

2.4.2 TRACK SELECTION SCREEN

This screen provide the UI elements to allow the player to dedice in which track he wants to race, it features three buttons one to delete the current track, one to select it and lastly a button to access the track editor screen of the game.



2.4.2.1 *SAVING/LOADING TRACKS*

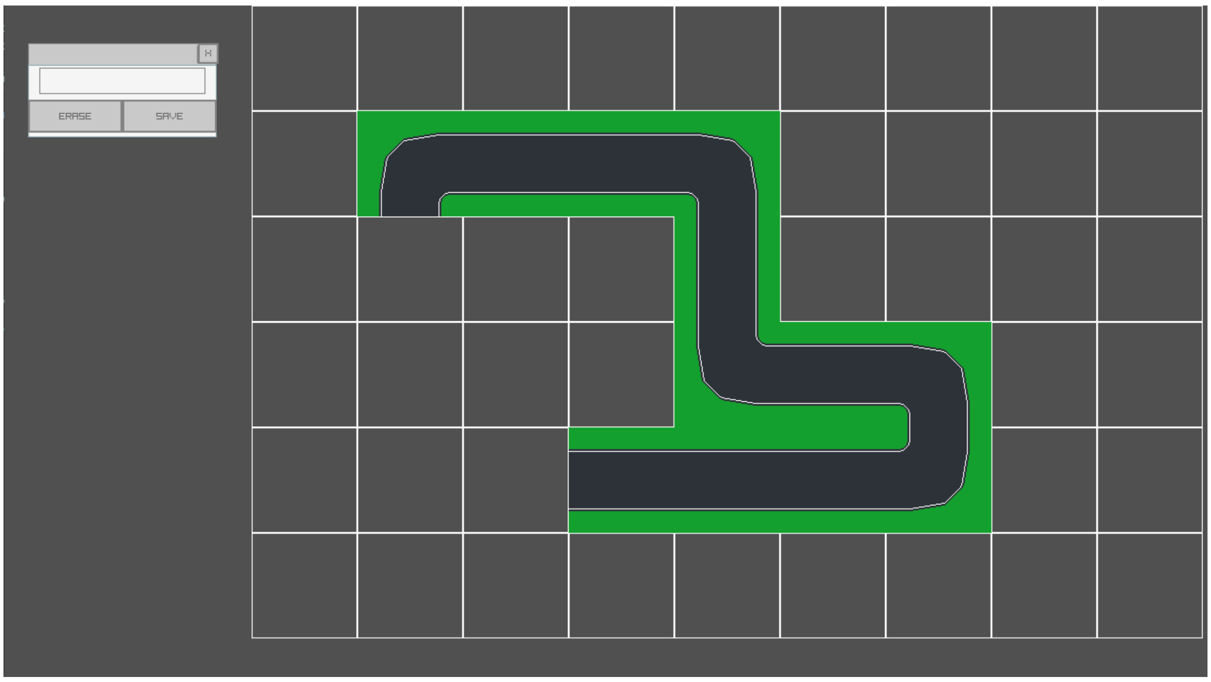
To handle saving and loading the tracks created by the user in different runs the game uses a binary file and the class TrackManager. TrackManager is a static class meaning it does not require creating an instance of it to use its methods. The variable trackFileName defines the name of the binary file where the track are saved. The SaveTracks method gets passed the current list of tracks in the game, it opens the file named "Tracks.bin" in create mode meaning that it open it **if** it exists or create a new file **if** it doesn't, the method uses a binary formatter to serialize the trackList into a format that can be stored in a binary file and saves the list of tracks in the binary file so it can be accessed after the games it's closed and lastly it returns the trackList. So we stored the current tracks in the Binary file.

```
STATIC CLASS TrackManager()
  CONST STRING TrackFileName ← "Tracks.bin"
  FUNCTION SaveTracks(LIST<Track> tracklist) RETURNS LIST<Track>
    USING(STREAM stream ← File.Open(trackFileName, Filemode.Create))
    VAR bformatter ← new System.Runtime.Serialization.Formatters.BinaryFormatter()
    Bformatter.Serialize(stream, tracklist)
    END USING
    RETURN trackList
  END FUNCTION
  FUNCTION LoadTracks(LIST<Track> trackList) RETURNS LIST<Track>
    USING(STREAM stream ← File.Open(trackFileName, Filemode.Open))
    VAR bformatter ← new System.Runtime.Serialization.Formatters.BinaryFormatter()
    trackList← Bformatter.Deserialize(stream)
    END USING
    RETURN trackList
  END FUNCTION
END CLASS
```

The LoadTracks method gets passed a list of tracks called trackList, it opens the file named "Tracks.bin" and reads it, the method uses a binary formatter to deserialize the data from the file and storing the data read in the passed list of tracks trackList, finally the methods returns the trackList parameter. So with this method we retrieved the saved track in the binary file and store them in a list of tracks.

2.4.3 TRACK EDITOR SCREEN

The track editor screen provides the user with the possibility of creating, editing, validating and saving tracks within the game, it manages the grid layout, the user interaction with the grid to create the track using tiles and track validation to ensure that only a track that can be used would be saved.



2.4.3.1 TRACK VALIDATION ALGORITHM

A track to be called valid needs to have some main characteristics, it needs to be connected not overlapping closed and each piece has to be linked and rotated in the right direction. To implement this validation in this game I used these methods in the TrackEditorScreen Class and implementations of the Cell and Waypoint class. A Cell defines the 7 different types of tile used to form a track by using his rotation,road and startingLine variables, rotation indicates the 90 degree rotation of the tile and the road variable indicates if its either a curve, a straight road or a grass tile, instead the startingLine Boolean indicates when its true that the cell is the starting line for the race track.

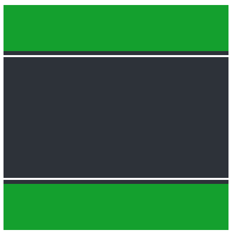
```
CLASS Cell
   STATIC INT width ← 112
   INT height ← 112
   BOOL traversed ← FALSE
   INT rotation ← 0
   INT road ← 0
   BOOL startingLine ← FALSE
ENDCLASS
```

CELL TYPES:

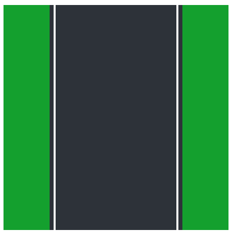
If Cell.road = 0 the cell is a grass tile

If Cell.road = 1 the cell is a straight road tile and based on his rotation it will look like one of this 2 options.

Cell.rotation = 0

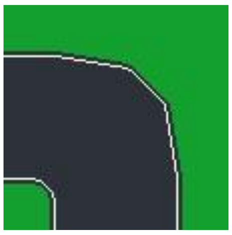


Cell.rotation = 1

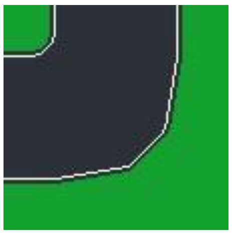


If Cell.road = 2 the cell is a curve tile and based on his rotation will look like one of this 4 options.

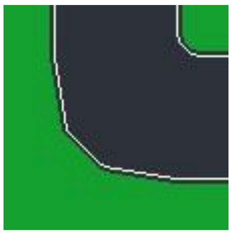
Cell.rotation = 0



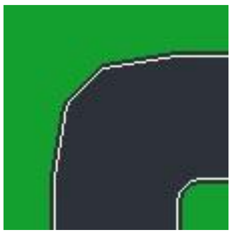
Cell.rotation = 1



Cell.rotation = 2



Cell.rotation = 3



The IsTrackValid() method checks is the first part of the validation process, it checks that the user created track is valid by iterating through the grid cells, for each cell with a road tile it checks if it has a valid connection with neighboring cells based on road types and rotations values. It returns true if the entire track is valid and connected, false otherwise, unluckily this is not enough for our validation process since it wouldn't classify as an error the creation by the user of two distinct tracks that are both closed and connected but not between each others (example given in figure 16) to fix this problem the games implement two other methods for validation scope IsTrackOnly() and Traverse(). in the function IsTrackValid() are instantiated 10 variables, validNeighb that keeps track of valid neighbours for the current cell, the Booleans validCell and validLine that respectively indicates if the track is valid so far and if the current row is already been validated, i and z that are used to loop through the grid and we also instantiate 5 Cell variables, the current one and other 4 that correspond to the 4 neighbouring grid cells in up,down,left and right directions.

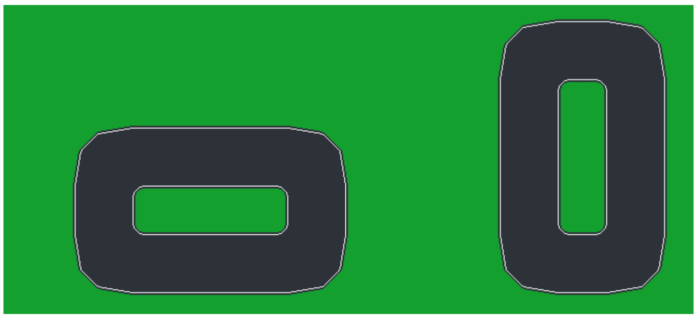


Figure 16 Not valid track that would be detected as valid

```
CLASS TrackEditorScreen
FUNCTION IsTrackValid()
    INT validNeighb ← 0
    BOOL validCell ← true
    BOOL validLine ← false
```

```
CELL upNeighb ← null
CELL bottNeighb ← null
CELL rightNeighb ← null
CELL leftNeighb ← null
CELL currNeighb ← null
INT i ← 0
INT z ← 0
```

After declaring the variables we start each cell in the grid (i for rows, z for columns). It retrieves the current cell (currNeighb) from the Grid Array,It checks if the current cell is a road tile or a grass tile. If its grass to the next cell. If the current cell is a road, it sets validLine to true, based on the current cell's road and rotation, it checks the validity of connections with neighboring cells,It retrieves neighboring cells and for each valid neighbor, it checks if the neighbor's road type and rotation allow for a valid connection according to the type of road tiles, If a valid connection is found, it increments the number of valid neighbours for the current cell incrementing validNeighb. After checking for the valid neighbors, it verifies validNeighb. A valid track cell (that is not a grass tile)should have exactly two valid connections ,If validLine is true and validNeighb is not equal to 2, it sets validCell to false, indicating that the track is not valid.After iterating through the entire grid, the function returns the final value of validCell, If validCell remains true, the entire grid represents a valid track.

```
WHILE i < Grid.height AND validCell != false
  validLine ← false

  WHILE z < Grid.width
    currNeighb ← Grid.GridArray[z, i]
    upNeighb ← new Cell()
    bottNeighb ← new Cell()
    rightNeighb ← new Cell()
    leftNeighb ← new Cell()
    validNeighb ← 0

    IF z != 0 THEN
      leftNeighb ← Grid.GridArray[z - 1, i]
    ENDIF
    IF z != 8 THEN
      rightNeighb ← Grid.GridArray[z + 1, i]
    ENDIF
    IF i != 0 THEN
      upNeighb ← Grid.GridArray[z, i - 1]
    ENDIF
    IF i != 5 THEN
      bottNeighb ← Grid.GridArray[z, i + 1]
    ENDIF
    IF currNeighb.Road != 0 THEN
      validLine ← true

    IF currNeighb.Road == 1 THEN
```

```
            IF (leftNeighb.Road == 1 AND leftNeighb.Rotation == 0) OR (leftNeighb.Road == 2 AND currNeighb.Rotation == 0 AND (leftNeighb.Rotation == 3 OR leftNeighb.Rotation == 2)) THEN
                validNeighb++
            ENDIF
            IF (rightNeighb.Road == 1 AND rightNeighb.Rotation == 0) OR (rightNeighb.Road == 2 AND currNeighb.Rotation == 0 AND (rightNeighb.Rotation == 0 OR rightNeighb.Rotation == 1)) THEN
                validNeighb++
            ENDIF
            IF (upNeighb.Road == 1 AND upNeighb.Rotation == 1) OR (upNeighb.Road == 2 AND currNeighb.Rotation == 1 AND (upNeighb.Rotation == 0 OR upNeighb.Rotation == 3)) THEN
                validNeighb++
            ENDIF
            IF (bottNeighb.Road == 1 AND bottNeighb.Rotation == 1) OR (bottNeighb.Road == 2 AND currNeighb.Rotation == 1 AND (bottNeighb.Rotation == 1 OR bottNeighb.Rotation == 2)) THEN
                validNeighb++
            ENDIF


        ELSE IF currNeighb.Road == 2 THEN

            IF currNeighb.Rotation == 0 THEN
                IF (bottNeighb.Road == 1 AND bottNeighb.Rotation == 1) OR (bottNeighb.Road == 2 AND (bottNeighb.Rotation == 1 OR bottNeighb.Rotation == 2)) THEN
                    validNeighb++
                ENDIF
                IF (leftNeighb.Road == 1 AND leftNeighb.Rotation == 0) OR (leftNeighb.Road == 2 AND (leftNeighb.Rotation == 2 OR leftNeighb.Rotation == 3)) THEN
                    validNeighb++
                ENDIF
            ELSE IF currNeighb.Rotation == 1 THEN
                IF (leftNeighb.Road == 1 AND leftNeighb.Rotation == 0) OR (leftNeighb.Road == 2 AND (leftNeighb.Rotation == 2 OR leftNeighb.Rotation == 3)) THEN
                    validNeighb++
                ENDIF
                IF (upNeighb.Road == 1 AND upNeighb.Rotation == 1) OR (upNeighb.Road == 2 AND (upNeighb.Rotation == 0 OR upNeighb.Rotation == 3)) THEN
                    validNeighb++
                ENDIF
            ELSE IF currNeighb.Rotation == 2 THEN
                IF (upNeighb.Road == 1 AND upNeighb.Rotation == 1) OR (upNeighb.Road == 2 AND (upNeighb.Rotation == 0 OR upNeighb.Rotation == 3)) THEN
                    validNeighb++
                ENDIF
                IF (rightNeighb.Road == 1 AND rightNeighb.Rotation == 0) OR (rightNeighb.Road == 2 AND (rightNeighb.Rotation == 0 OR rightNeighb.Rotation == 1)) THEN
                    validNeighb++
                ENDIF
            ELSE IF currNeighb.Rotation == 3 THEN
```

```
                IF (bottNeighb.Road == 1 AND bottNeighb.Rotation == 1) OR (bottNeighb.Road == 2 AND (bottNeighb.Rotation == 1 OR bottNeighb.Rotation == 2)) THEN
                    validNeighb++
                ENDIF
                IF (rightNeighb.Road == 1 AND rightNeighb.Rotation == 0) OR (rightNeighb.Road == 2 AND (rightNeighb.Rotation == 0 OR rightNeighb.Rotation == 1)) THEN
                    validNeighb++
                ENDIF
              ENDIF
            ENDIF

            IF validNeighb != 2 AND validLine == true THEN
              validCell ← false
            ENDIF
          ENDIF

          z++
        ENDWHILE
        z ← 0
        i++
      ENDWHILE

      RETURN validCell
    ENDFUNCTION
```

The Traverse method its part of the validation process, this procedure takes as input a CORDINATE object that consists in the position of the starting cell of the grid, it initially instantiate various variables, the boolean trackTrav that will be used to track if the algorithm reaches the end of the track, 5 Cell objects curCell that will represent the current cell and the other 4 (upNeighb, bottNeighb, rightNeighb, leftNeighb) that are the neighboring cells, and a new Waypoint object is created with the current coordinates and added to the waypoints list that will be the path of the track.

```
CLASS TrackEditorScreen
LIST of Waypoint Waypoints

PROCEDURE Traverse(CORDINATE cordinate)
  BOOL trackTrav ← false
  BOOL isStraight ← false
  CELL currCell ← Grid.GridArray[cordinate.x, cordinate.y]
  CELL upNeighb ← null
  CELL bottNeighb ← null
  CELL rightNeighb ← null
  CELL leftNeighb ← null
  WAYPOINT currWaipoint ← new Waypoint()

  currWaipoint.x ← cordinate.x
  currWaipoint.y ← cordinate.y
  Waypoints.Add(currWaipoint)
```

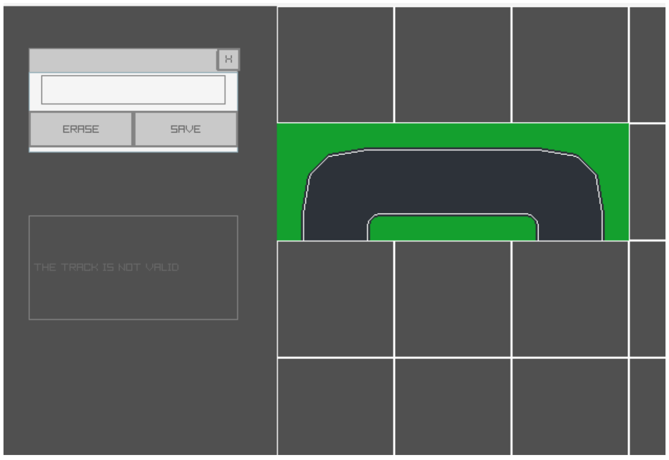
The core of this procedure consists in a recursive operation, it checks the current cell road and rotation value. Based on these values, it checks for valid neighboring cells in the 4s directions (up, down, left, right) based on the rotation it validates that are compatible road tiles. It then performs the following actions for each valid neighbor, it sets the current cell as Traversed,if the neighbor hasn't been traversed (Traversed 🡐 false), it recursively calls Traverse with the neighbor's coordinates, so exploring that path of the track graph furtherly. If none of the valid neighbors haven't been traversed yet, it sets trackTrav Boolean to true, indicating that the end has been reached on the path.

```
IF currCell.Road == 1 AND currCell.Rotation == 0 THEN
  IF cordinate.x > 0 THEN
    leftNeighb 🡐 Grid.GridArray[cordinate.x - 1, cordinate.y]
  ENDIF
  IF cordinate.x < 8 THEN
    rightNeighb 🡐 Grid.GridArray[cordinate.x + 1, cordinate.y]
  ENDIF

  currCell.Traversed 🡐 true

  IF rightNeighb != null AND rightNeighb.Traversed == false THEN
    Traverse(new Cordinate(cordinate.x + 1, cordinate.y))
  ELSEIF leftNeighb != null AND leftNeighb.Traversed == false THEN
    Traverse(new Cordinate(cordinate.x - 1, cordinate.y))
  ELSE
    trackTrav 🡐 true
  ENDIF
ENDIF
ELSEIF currCell.Road == 1 AND currCell.Rotation == 1 THEN
  IF cordinate.y > 0 THEN
    upNeighb 🡐 Grid.GridArray[cordinate.x, cordinate.y - 1]
  ENDIF
  IF cordinate.y < 5 THEN
    bottNeighb 🡐 Grid.GridArray[cordinate.x, cordinate.y + 1]
  ENDIF

  currCell.Traversed 🡐 true

  IF upNeighb != null AND upNeighb.Traversed == false THEN
    Traverse(new Cordinate(cordinate.x, cordinate.y - 1))
  ELSEIF bottNeighb != null AND bottNeighb.Traversed == false THEN
    Traverse(new Cordinate(cordinate.x, cordinate.y + 1))
  ELSE
    trackTrav 🡐 true
  ENDIF
ENDELSEIF
ELSEIF currCell.Road == 2 AND currCell.Rotation == 0 THEN
  IF cordinate.x > 0 THEN
    leftNeighb 🡐 Grid.GridArray[cordinate.x - 1, cordinate.y]
  ENDIF
  IF cordinate.y < 5 THEN
    bottNeighb 🡐 Grid.GridArray[cordinate.x, cordinate.y + 1]
  ENDIF

  currCell.Traversed ← true

  IF leftNeighb != null AND leftNeighb.Traversed == false THEN
    Traverse(new Cordinate(cordinate.x - 1, cordinate.y))
  ELSEIF bottNeighb != null AND bottNeighb.Traversed == false THEN
    Traverse(new Cordinate(cordinate.x, cordinate.y + 1))
  ELSE
    trackTrav ← true
  ENDIF
ENDELSIF
ELSEIF currCell.Road == 2 AND currCell.Rotation == 1 THEN
  IF cordinate.x > 0 THEN
    leftNeighb ← Grid.GridArray[cordinate.x - 1, cordinate.y]
  ENDIF
  IF cordinate.y > 0 THEN
    upNeighb ← Grid.GridArray[cordinate.x, cordinate.y - 1]
  ENDIF

  currCell.Traversed ← true

  IF leftNeighb != null AND leftNeighb.Traversed == false THEN
    Traverse(new Cordinate(cordinate.x - 1, cordinate.y))
  ELSEIF upNeighb != null AND upNeighb.Traversed == false THEN
    Traverse(new Cordinate(cordinate.x, cordinate.y - 1))
  ELSE
    trackTrav ← true
  ENDIF
ENDELSIF
ELSEIF currCell.Road == 2 AND currCell.Rotation == 2 THEN
  IF cordinate.x < 8 THEN
    rightNeighb ← Grid.GridArray[cordinate.x + 1, cordinate.y]
  ENDIF
  IF cordinate.y > 0 THEN
    upNeighb ← Grid.GridArray[cordinate.x, cordinate.y - 1]
  ENDIF

  currCell.Traversed ← true

  IF rightNeighb != null AND rightNeighb.Traversed == false THEN
    Traverse(new Cordinate(cordinate.x + 1, cordinate.y))
  ELSEIF upNeighb != null AND upNeighb.Traversed == false THEN
    Traverse(new Cordinate(cordinate.x, cordinate.y - 1))
  ELSE
    trackTrav ← true
  ENDIF
ENDELSEIF
ELSEIF currCell.Road == 2 AND currCell.Rotation == 3 THEN
  IF cordinate.x < 8 THEN
    rightNeighb ← Grid.GridArray[cordinate.x + 1, cordinate.y]
  ENDIF
  IF cordinate.y < 5 THEN
    bottNeighb ← Grid.GridArray[cordinate.x, cordinate.y + 1]
  ENDIF

  currCell.Traversed ← true

  IF rightNeighb != null AND rightNeighb.Traversed == false THEN
    Traverse(new Cordinate(cordinate.x + 1, cordinate.y))
  ELSEIF bottNeighb != null AND bottNeighb.Traversed == false THEN
    Traverse(new Cordinate(cordinate.x, cordinate.y + 1))
  ELSE
    trackTrav ← true
  ENDIF

ENDIF
ENDELSIF
ENDPROCEDURE
```

IsTrackOnly() is the last method needed to ensure that the track is completes the work of Traversed() method by checking that all the road cell has been traversed by this method and so ensuring that the we are not encountering a situation similar to Figure 16 where there are two completed tracks that are not connected. It instantiates 3 variables x,y and valid the 2 integers x,y are used to iterate through the grid instead valid is the Boolean(setting it at the start equal to false)that is returned by this function as true if the track is valid and false if its not. The method iterates trough the grid it checks if the current cell if the current cell is a grass tile (Cell.Road = 0) if not it checks if the current cell has been traversed and if it has not it sets valid as false meaning that the track is not valid since more tiles than the track traversed by the Traverse method.

```
FUNCTION BOOL IsTrackOnly()
  INT x ← 0
  INT y ← 0
  BOOL valid ← true

  WHILE y < 6 AND valid != false
    x ← 0
    WHILE x < 9
      IF Grid.GridArray[x, y].Road != 0 THEN
        IF Grid.GridArray[x, y].Traversed == false THEN
          valid ← false
        ENDIF
      ENDIF
      x++
    ENDWHILE
    y++
  ENDWHILE
  RETURN valid
ENDFUNCTION
```

The validation algorithm is used in the code and these methods are called when the save button is pressed inside the Draw() method in the TrackEditorScreen Class. If pressed it calls it find the first cell with a straight road that will be the starting point, if found it calls Traverse to explore the track from the starting point. Then it calls IsTrackValid() to check if the explored track is structurally valid and IsTrackOnly() to check if the user didn't create two or more closed and not connected tracks(such as in Figure 16), and if both validations pass: it define the starting point on the grid as equal to the values of the first straight road found before it creates a new Track object with the grid, track name, and the track waypoints, it adds this item to the trackList and saves this list into the binary file using the track manager. It closes the editing screen using m_gameScreen.PopScreen. Instead if validation fails it sets ErrorMessage to true to display an error message(Figure 17)to the user to tell him that the track validation hasn't been successful so is not possible to save the track.



THE TRACK IS NOT VALID

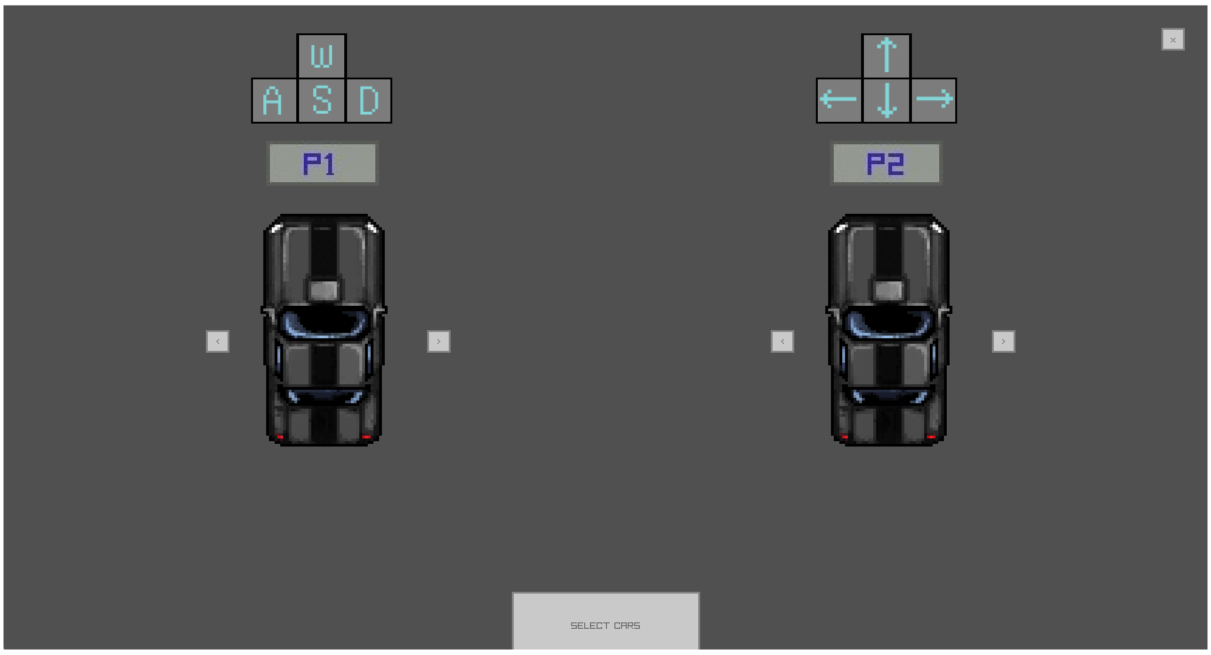
Figure 17 Error message

```
CLASS TrackEditorScreen
FUNCTION Draw()
  ……..
  ELSE IF RayGui.GuiButton(new Rectangle(125, 100, 100, 35), "SAVE") == true THEN

    cordinate = firstStraightCell()
      IF cordinate.x != -1 AND cordinate.y != -1 THEN
      Traverse(cordinate)
      IF IsTrackValid() == true AND IsTrackOnly() == true THEN
        Grid.Fill = true
        Grid.SetStartingPoint()
        Track newTrack = new Track(Grid, trackName, Waypoints)
        TrackList.Add(newTrack)
        TrackManager.SaveTracks(TrackList)
        m_gameScreen.PopScreen()
        TrackSelectionScreen.currentTrack++
      ELSE
        ErrorMessage = true
      ENDIF
    ELSE
      ErrorMessage = true
    ENDIF
  ENDIF

  IF ErrorMessage == true THEN
    bool editmode = false

    IF RayGui.GuiTextBox(new Rectangle(25, 200, 200, 100), "THE TRACK IS NOT VALID", 25,
editmode) THEN
```

```
        Grid.Erase()
        ErrorMessage = false
      ENDIF
    ENDIF

  ENDFUNCTION
```

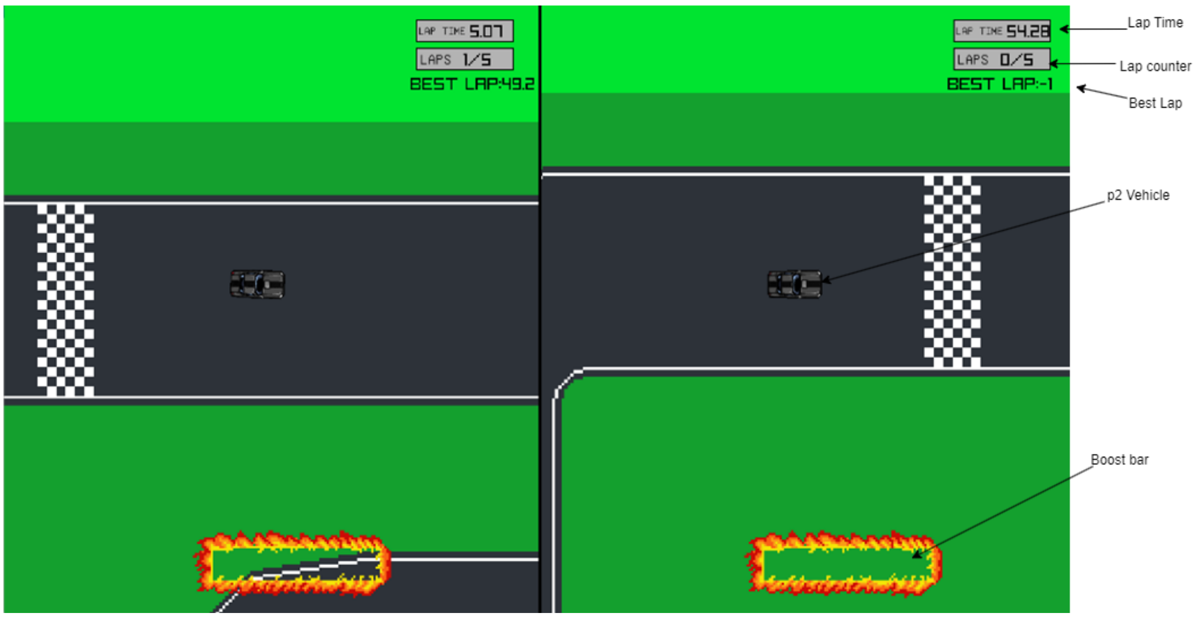
2.4.4 CAR SELECTION SCREEN

The car selection screen provides the user with The UI elements to select the 2 cars to use in the race and decide whether to play against another player or a bot. it features 4 arrows to scroll through the first and second player cars, it shows the 2 cars textures, whether if are player controlled or "cpu controlled" and the keys to control them and 2 buttons "x" to go back to the previous screen and "SELECT CARS" to select the current cars that the screen is showing.



2.4.5 GAMEPLAY SCREEN

The gameplay screen provides a split screen camera view to permit the two playe=l/rs, either both users or the user and a bot, to play in different part of the screen with the left side camera following the first and the right one the second player, it provides the users HUD with various data, the boost bar in the bottom part of both sides the splitted screen help the player know the amount of boost remaining, in the top right corners of both sides the lap time indicating the time of the current player lap, the lap counter indicating the numbers of lap completed by that player and the best lap the player has done in the race are provided.



2.4.5.1 Lap validation algorithm

This algorithm is used to check that the player completes the track fully each lap preventing the game to register as a valid lap one where the player used shortcuts and tried to exploit the game, to guarantee a working validation the game implements checkpoints and a checkpoint manager. To do so firstly the games implements a checkpoint class that defines a checkpoint by declaring CheckRect that is the rectangle object defining the area where the car needs to pass to trigger the checkpoint, and it initializes the checkpoint with a passed checkRect value in the constructor.

```
CLASS Checkpoint
    RECTANGLE checkRect

    FUNCTION Checkpoint(RECTANGLE checkRect)
        this.CheckRect ← checkRect
    ENDFUNCTION
ENDCLASS
```

The checkpoint manager, manages checkpoints on the race track for the two players and validates laps based on checkpoints passed. To do it uses various data structures, a list of checkpoints checkList that contains all the checkpoints objects of the race track, 2 Queues (p1Queue and p2Queue) storing respectively upcoming checkpoints for Player 1 and Player 2 and the int p1Lapsleft and P2Lapsleft that keep track of the remaining laps for each player in order to conclude the race. In the constructor we set p1Lapsleft and p2Lapsleft to the passed in the constructor number of total laps in order to finish the race, and we loop through each checkpoint in checkpList and we enqueue them to both P1Queue and P2Queue but since to complete a lap the player needs to pass the starting line twice we enqueue another time the starting line checkpoint at the end of both queues.

```
CLASS CheckpointManager
    Queue<Checkpoint> p1Queue
    Queue<Checkpoint> p2Queue
    List<Checkpoint> checkpList
    int p1Lapsleft
    int p2Lapsleft
    FUNCTION CheckpointManager(List<Checkpoint> checkps, INT lapsNumber)
        P1Lapsleft ← lapsNumber
        P2Lapsleft ← lapsNumber
        CheckpList ← checkps
        FOR i ← 0 TO CheckpList.Count() - 1
            P1Queue.Enqueue(CheckpList.ElementAt(i))
            P2Queue.Enqueue(CheckpList.ElementAt(i))
        ENDFOR
        P1Queue.Enqueue(CheckpList.ElementAt(0))
        P2Queue.Enqueue(CheckpList.ElementAt(0))
    ENDFUNCTION
```

p1IsLapValid() and p2IsLapValid() checks if the respective player's queue is empty (meaning all checkpoints have been passed).If valid, decrements remaining laps since one has been completed and re-enqueues all checkpoints for the next lap. And returns a boolean indicating lap validity. p1CheckpointPassed() and p2CheckpointPassed() dequeues the first checkpoint from the respective player's queue when a player passes a checkpoint.

```
FUNCTION p1IsLapValid() RETURNS BOOL
        IsValid ← false
        IF P1Queue.Count == 0 THEN
            IsValid ← true
            P1Lapsleft--
            FOR i ← 0 TO CheckpList.Count() - 1
                P1Queue.Enqueue(CheckpList.ElementAt(i))
            ENDFOR
        ENDIF
        RETURN IsValid
    ENDFUNCTION

    FUNCTION p2IsLapValid() RETURNS BOOL
        IsValid ← false
        IF P2Queue.Count == 0 THEN
            P2Lapsleft--
            IsValid ← true
            FOR i ← 0 TO CheckpList.Count() - 1
                P2Queue.Enqueue(CheckpList.ElementAt(i))
            ENDFOR
        ENDIF
        RETURN IsValid
    ENDFUNCTION

    FUNCTION p1CheckpointPassed()
        P1Queue.Dequeue()
    ENDFUNCTION

    FUNCTION p2CheckpointPassed()
        P2Queue.Dequeue()
    ENDFUNCTION
ENDCLASS
```

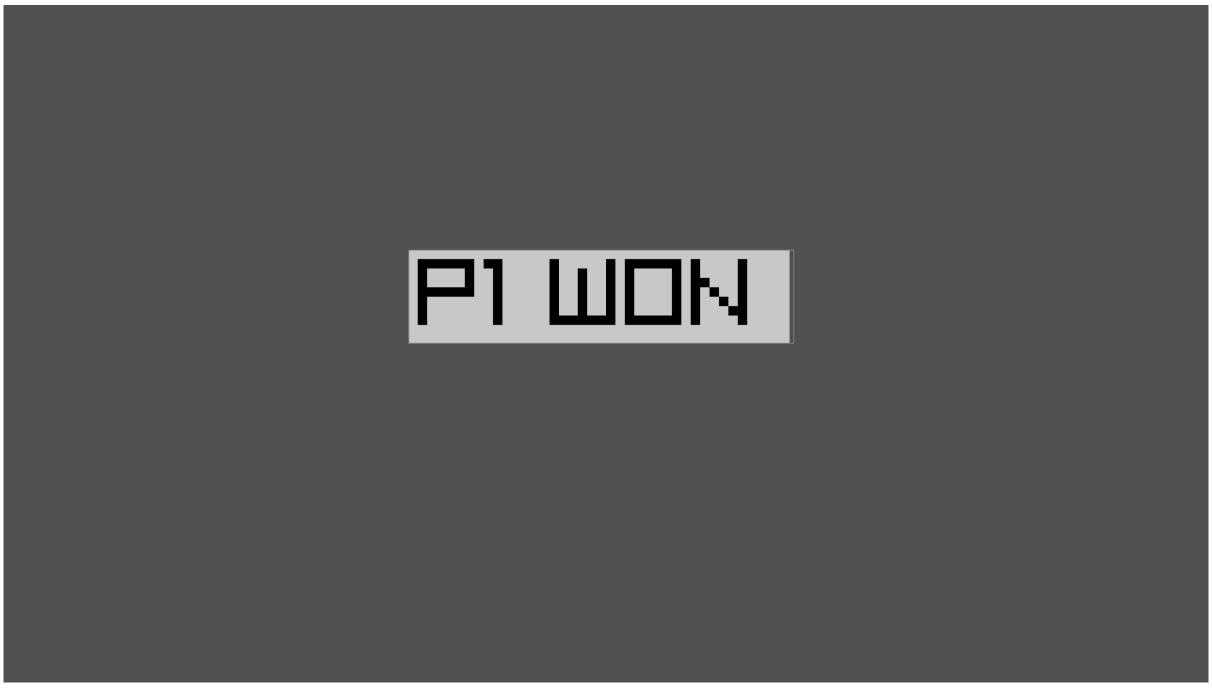
The CheckpointManager is used in the Gamescreen class to complete the laps validation process using the LapCollisions() method. Determines the hitbox Rectangle for the second player using p2Bot.CarRect if playing against a CPU or using p2Car.PlayerRect. After for player 1 and 2 it checks If there are checkpoints remaining and if the car collides with the first checkpoint in the queue(that corresponds to the next one that the player would encounter if percurring the track in the right order), In case both these conditions are satisfied it calls the checkpoint manager

P1CheckpointPassed() or P2CheckpointPassed() (based on which player passed the checkpoint) method so dequeueing the first element of the corresponding player checkpoint queue. It also handles lap completion and winning conditions, if one of the two players car collides with the start/finish line and the lap is valid(this is checked by using the checkpoint manager p1IsLapValid() or p2IsLapValid() method), it increments their lap count (P1lapsdone++ or P2lapsdone++), updates the best lap time if the one done in this lap is better than the previous register and restarts the timer for the next lap. But if at the moment of the player car collision the laps were already all completed (so meaning that p1lapsdone in case it's the first player car or p2lapsdone in case it's the second player its equal to Laps that corresponds to the total numbers of laps of the race) the game will show another screen displaying the name of the winner of the race.

```
FUNCTION LapCollisions()
  IF playCPU THEN
    rect2 ← p2Bot.CarRect
  ELSE
    rect2 ← p2Car.PlayerRect
  ENDIF
  IF checkpointManager.P1Queue.Count() != 0 AND
Raylib.CheckCollisionRecs(gameCar.PlayerRect, checkpointManager.P1Queue.First().CheckRect)
THEN
    checkpointManager.p1CheckpointPassed()
  ENDIF
  IF checkpointManager.P2Queue.Count() != 0 AND Raylib.CheckCollisionRecs(rect2,
checkpointManager.P2Queue.First().CheckRect) THEN
    checkpointManager.p2CheckpointPassed()
  ENDIF

  IF P2lapsdone == Laps AND Raylib.CheckCollisionRecs(rect2, startcellRect) AND
checkpointManager.p2IsLapValid() == true THEN
    m_screenManager.PushScreen(new WinnerScreen("P2", m_screenManager))
    Raylib.SetWindowSize(1280, 720)
  ELSEIF P1lapsdone == Laps AND Raylib.CheckCollisionRecs(gameCar.PlayerRect, startcellRect)
AND checkpointManager.p1IsLapValid() == true THEN
    m_screenManager.PushScreen(new WinnerScreen("P1", m_screenManager))
    Raylib.SetWindowSize(1280, 720)
  ELSEIF P2lapsdone != Laps AND Raylib.CheckCollisionRecs(rect2, startcellRect) AND
checkpointManager.p2IsLapValid() == true THEN
    P2lapsdone++
    IF p2currentTime <= p2bestLap OR p2bestLap < 0 THEN
      p2bestLap ← p2currentTime
    ENDIF
    p2startingTime ← p1currentTime + p1startingTime
  ELSEIF P1lapsdone != Laps AND Raylib.CheckCollisionRecs(gameCar.PlayerRect, startcellRect)
AND checkpointManager.p1IsLapValid() == true THEN
    P1lapsdone++
    IF p1currentTime <= p1bestLap OR p1bestLap < 0 THEN
      p1bestLap ← p1currentTime
    ENDIF
    p1startingTime ← p1currentTime + p1startingTime
  ENDIF
ENDFUNCTION
```

2.4.6 WINNER SCREEN

The winner screen displays in the middle of the screen the player that as won the race, if the grey button that shows the winner is pressed it will bring the user back to the track selection screen.



2.5 MAIN DATA STRUCTURE

2.5.1 Queues

Queues are used in the Checkpointmanager class for storing the upcoming checkpoint for Player1 and Player 2 and are the core of the lap validation algorithm of the game. This by dequeing all the checkpoint in which the car passed in the current lap and checking if the queues are empty, and so all the track checkpoints have been percurred, when the cars traverse the starting/finishing line.

2.5.2 2D array

A bidimensional array of Cell is used in the Grid class to represent the 9x6 race track grid providing a structured way to store and manage the track informations relatives to each chell. It allows for efficient drawing of the track based on the attributes of each cell in the 2D array and facilitates the user interaction for editing and creating the track layout in the track editor screen.

2.5.3 File structure

In the solution is used a single binary file named “Tracks.bin” that is used to store a list containing all the tracks in the game. The file is loaded and written in the TrackManager class using a binary formatter, When saving tracks, the SaveTracks method serializes the entire List<Track> object containing all the tracks into the binary file. Instead when loading tracks, the LoadTracks method deserializes the content of the binary file back into a List<Track> object. This object is then assigned to the trackList parameter passed to the method and returned.

2.5.4 2D Vectors

2D vectors are extensively used throughout the solution to represent various properties, positions on the screen, forces and perform calculations. For example in the Seek function in the CarAi class The desired car velocity towards the target waypoint is calculated by subtracting the car's position vector from the target waypoint vector and the steering force is calculated by subtracting the car's current velocity vector from the desired velocity vector.

3 IMPLEMENTATION

Technical Skill	Description	Class reference
Queue operations	Queues are used to manage the checkpoints percurred by the two players every lap and so for convalidating the laps	CheckpointManager.cs
List Operations/Stack operations	List is used to manage the game screens that are drawn and updated and constantly used In other occasion throughout the solution. A List is implemented in the GameScreenManager class with a LIFO stack behaviour.	GameScreenManager.cs
Multidimensional arrays	A 2d array of Cells is used to represent the track cell grid	Grid.cs
Recursive algorithms	Used in the Traverse() function to traverse the track grid for validation scope	TrackEditorScreen.cs
Mathematical calculations	Mathematical calculations are present all throughout the solution for example linear interpolation	e.g Star.cs in the Map() function
File Handling	Writing and reading a binary file to store and use the game tracks	TrackManager.cs
Complex object orientated programming – inheritance	Game screens inherit from the IGameScreen interface, the GameScreenManager class inherits from the IGameScreenManager interface, and the PlayerCar class inherits from the Vehicle class	PlayerCar.cs SplashScreen TrackselectionScreen.cs TrackEditorScreen.cs CarSelectionScreen GameScreen.cs WinnerScreen.cs GameScreenManager.cs
Complex object orientated programming – interfaces	Gives a template for the screen manager and the game screens in the solution.	ScreenInterfaces.cs
Graph Traversal	Traverse the track cells as a graph as a part of the user created tracks validation process.	TrackEditorScreen.cs in the Traverse() function

3.1 PROGRAM.CS

```
using NEA;
using NEA.Screens;

using Raylib_CsLo;
using System.Numerics;
using System.Threading.Tasks;

namespace NEA
{
    public static class Program
    {


        public static Vehicle vehicle;
        public static Grid grid;


        public static Texture carTexture;
        public static Texture carTexture2;
        public static Texture carTexture3;
        public static Texture selectionP1;
        public static Texture selectionP2;
        public static Texture selectionCPU;
        public static Color backgroundColor = Raylib.DARKGRAY;
        public static Texture boostFrame;
        public static Texture Bush1;
        public static Texture Bush2;
        public static Texture Bush3;
        public static Texture Tree;
        public static Texture lapTime;
        public static Texture Laps;
        public static Texture WASD;
        public static Texture Arrows;
        public static Texture CARZ;




        public static int screenwidth = 1280;
        public static int screenheight = 720;


        public static IGameScreenManager m_screenManager = new
GameScreenManager();




        public static async Task Main(string[] args)
        {
            //initialize the screen
            Raylib.InitWindow(screenwidth, screenheight, "Hello, Raylib-CsLo");
            //Load all the game texture
            Grid.curveTexture = Raylib.LoadTexture("/Users/39351/OneDrive -
rendcomb.gloucs.sch.uk/Computing/NEA/NEA/bin/Debug/net6.0/Assets/Track/curve
1.png");
            Grid.roadTexture = Raylib.LoadTexture("/Users/39351/OneDrive -
rendcomb.gloucs.sch.uk/Computing/NEA/NEA/bin/Debug/net6.0/Assets/Track/straight
1.png");
```

```
            Grid.grassTexture = Raylib.LoadTexture("/Users/39351/OneDrive -
rendcomb.gloucs.sch.uk/Computing/NEA/NEA/bin/Debug/net6.0/Assets/Track/grass.png"
);
            Grid.startingLine = Raylib.LoadTexture("/Users/39351/OneDrive -
rendcomb.gloucs.sch.uk/Computing/NEA/NEA/bin/Debug/net6.0/Assets/Track/startingLi
ne.png");
            carTexture = Raylib.LoadTexture("/Users/39351/OneDrive -
rendcomb.gloucs.sch.uk/Computing/NEA/NEA/bin/Debug/net6.0/Assets/Cars/BlackCar.pn
g");
            carTexture2 = Raylib.LoadTexture("/Users/39351/OneDrive -
rendcomb.gloucs.sch.uk/Computing/NEA/NEA/bin/Debug/net6.0/Assets/Cars/SportCar.pn
g");
            carTexture3 = Raylib.LoadTexture("/Users/39351/OneDrive -
rendcomb.gloucs.sch.uk/Computing/NEA/NEA/bin/Debug/net6.0/Assets/Cars/WhiteCar.pn
g");
            selectionP1 = Raylib.LoadTexture("/Users/39351/OneDrive -
rendcomb.gloucs.sch.uk/Computing/NEA/NEA/bin/Debug/net6.0/Assets/P1.png");
            selectionP2 = Raylib.LoadTexture("/Users/39351/OneDrive -
rendcomb.gloucs.sch.uk/Computing/NEA/NEA/bin/Debug/net6.0/Assets/P2.png");
            selectionCPU = Raylib.LoadTexture("/Users/39351/OneDrive -
rendcomb.gloucs.sch.uk/Computing/NEA/NEA/bin/Debug/net6.0/Assets/CPU.png");
            boostFrame = Raylib.LoadTexture("/Users/39351/OneDrive -
rendcomb.gloucs.sch.uk/Computing/NEA/NEA/bin/Debug/net6.0/Assets/Frame.png");
            Bush1 = Raylib.LoadTexture("/Users/39351/OneDrive -
rendcomb.gloucs.sch.uk/Computing/NEA/NEA/bin/Debug/net6.0/Assets/Bush1.png");
            Bush2 = Raylib.LoadTexture("/Users/39351/OneDrive -
rendcomb.gloucs.sch.uk/Computing/NEA/NEA/bin/Debug/net6.0/Assets/Bush2.png");
            Bush3 = Raylib.LoadTexture("/Users/39351/OneDrive -
rendcomb.gloucs.sch.uk/Computing/NEA/NEA/bin/Debug/net6.0/Assets/Bush3.png");
            Tree = Raylib.LoadTexture("/Users/39351/OneDrive -
rendcomb.gloucs.sch.uk/Computing/NEA/NEA/bin/Debug/net6.0/Assets/Tree.png");
            lapTime = Raylib.LoadTexture("/Users/39351/OneDrive -
rendcomb.gloucs.sch.uk/Computing/NEA/NEA/bin/Debug/net6.0/Assets/Laptime.png");
            Laps = Raylib.LoadTexture("/Users/39351/OneDrive -
rendcomb.gloucs.sch.uk/Computing/NEA/NEA/bin/Debug/net6.0/Assets/Laps.png");
            WASD = Raylib.LoadTexture("/Users/39351/OneDrive -
rendcomb.gloucs.sch.uk/Computing/NEA/NEA/bin/Debug/net6.0/Assets/WASD.png");
            Arrows = Raylib.LoadTexture("/Users/39351/OneDrive -
rendcomb.gloucs.sch.uk/Computing/NEA/NEA/bin/Debug/net6.0/Assets/Arrows.png");
            CARZ = Raylib.LoadTexture("/Users/39351/OneDrive -
rendcomb.gloucs.sch.uk/Computing/NEA/NEA/bin/Debug/net6.0/Assets/CARZ.png");


            grid = new Grid();

            Raylib.SetTargetFPS(60);

            m_screenManager.PushScreen(new SplashScreen(m_screenManager));


            // Main game loop
            while (!Raylib.WindowShouldClose()) // Detect window close button or
ESC key
            {

                Raylib.BeginDrawing();

                Raylib.ClearBackground(Raylib.DARKGRAY);
                Draw();

                Update();
                Raylib.EndDrawing();
            }
            Raylib.CloseWindow();
        }


        public static void Update()
        {
            m_screenManager.Update();
            m_screenManager.HandleInput();

            bool posoccupied = false;





        }

        public static void Draw()
        {
            m_screenManager.Draw();

        }
    }
}
```

3.2 CELL.CS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NEA
{
    [Serializable]
    public class Cell
    {

        public static int width = 112;
        private int height = 112;
        private bool traversed = false;
        private int rotation = 0;
        private int road = 0;
        private bool startingLine = false;


        public int Height { get => height; set => height = value; }
        public bool Traversed { get => traversed; set => traversed = value; }
        public int Rotation { get => rotation; set => rotation = value; }
        public int Road { get => road; set => road = value; }
        public bool StartingLine { get => startingLine; set => startingLine =
value; }
```

```
        public Cell()
        {

        }


    }
}
```

3.3 GRID.CS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Numerics;
using System.Reflection;
using System.Reflection.Metadata;
using System.Reflection.Metadata.Ecma335;
using System.Text;
using System.Threading.Tasks;
using Raylib_CsLo;
using SharpDX.DXGI;


namespace NEA
{

    [Serializable]
    public class Grid
    {

        public const int width = 9;
        public const int height = 6;
        private Cell[,] gridArray;
        public static Cell cell = new Cell();
        private bool fill = false;
        public static Texture curveTexture;
        public static Texture roadTexture;
        public static Texture grassTexture;
        public static Texture startingLine;

        public Cell[,] GridArray { get => gridArray; set => gridArray = value; }
        public bool Fill { get => fill; set => fill = value; }

        public Grid()
        {
            GridArray = new Cell[width, height];


            for (int i = 0; i < height; i++)
            {
                for (int z = 0; z < width; z++)
                {

                    GridArray[z,i] = new Cell();
                }
            }

        }
        public void Draw(int cellxy , int gridStartingX, float scale, int 
startLine)
        {
            int cellx = 0;
            int celly = 0;

            Vector2 roadCenter = new 
Vector2(roadTexture.width/2,roadTexture.height/2);



            for (int y = 0; y < height; y++)
            {

                for (int i = 0; i < width; i++)
                {
                    cellx += cellxy;
                    Raylib.DrawRectangleLinesEx(new Rectangle( startLine + cellx, 
celly, cellxy, cellxy), 1, Raylib.WHITE);
                }
                cellx = 0;
                celly += cellxy;

            }

            for (int i = 0; i < height; i++)
            {
                for (int z = 0; z < width; z++)
                {


                    if (GridArray[z, i].Road == 0 && Fill == true)
                    {
                        Raylib.DrawTexturePro(grassTexture, new Rectangle(0, 0, 
roadTexture.width, roadTexture.height), new Rectangle((z * cellxy + 
gridStartingX) + (roadCenter.X * scale), i * cellxy + (roadCenter.X * scale), 
roadTexture.width * scale, roadTexture.height * scale), roadCenter * scale, 
GridArray[z, i].Rotation * 90, Raylib.WHITE);
                    }



                    else if (GridArray[z,i].Road == 1)
                    {

                        if (GridArray[z, i].StartingLine == true)
                        {
                            Raylib.DrawTexturePro(startingLine, new Rectangle(0, 
0, roadTexture.width, roadTexture.height), new Rectangle((z * cellxy + 
gridStartingX) + (roadCenter.X * scale), i * cellxy + (roadCenter.X * scale), 
roadTexture.width * scale, roadTexture.height * scale), roadCenter * scale, 
GridArray[z, i].Rotation * 90, Raylib.WHITE);

                        }

                        else
                        {
                            // we use texture pro to set the origin of the 
rotation to the centre of the texutre
```

```
                            Raylib.DrawTexturePro(roadTexture, new Rectangle(0,
0, roadTexture.width, roadTexture.height), new Rectangle((z * cellxy +
gridStartingX) + (roadCenter.X * scale), i * cellxy + (roadCenter.X * scale),
roadTexture.width * scale, roadTexture.height * scale), roadCenter * scale,
GridArray[z, i].Rotation * 90, Raylib.WHITE);

                        }



                    }
                    else if (GridArray[z,i].Road == 2)
                    {
                        Raylib.DrawTexturePro(curveTexture, new Rectangle(0, 0,
curveTexture.width, curveTexture.height), new Rectangle((z * cellxy +
gridStartingX) + (roadCenter.X * scale), i * cellxy + (roadCenter.X * scale),
curveTexture.width * scale, curveTexture.height * scale), roadCenter * scale,
GridArray[z, i].Rotation * 90, Raylib.WHITE);
                    }


                }

            }
        }

        /// <summary>
        /// Checks which cell of the grid we pressed and changes it to a
different map tile according to which button we pressed
        /// </summary>
        public void IsCellPicked()
        {


            int mouseX = 0;
            int mouseY = 0;
            int cellxy = 112;


            // if the right mouse button is clicked a straight its placed, if the
cell picked is already a straight it will be rotated
            if (Raylib.IsMouseButtonPressed(MouseButton.MOUSE_BUTTON_RIGHT) ==
true)
            {

                mouseX = Raylib.GetMouseX();
                mouseY = Raylib.GetMouseY();

                if (mouseX >= 262 && mouseX <= 1270 && mouseY <= 672)
                {

                    // we find the element of the array corrispondent to this
cordinates in our grid and we set to 1 to indicate that we drew a road in that
cell
                    mouseX = (mouseX - 262) / cellxy;
                    mouseY = mouseY / cellxy;


                    if (GridArray[mouseX, mouseY].Road == 1)
                    {

                        if (GridArray[mouseX, mouseY].Rotation == 1)
                        {
                            GridArray[mouseX, mouseY].Rotation = 0;
                        }
                        else if (GridArray[mouseX, mouseY].Rotation == 0)
                        {
                            GridArray[mouseX, mouseY].Rotation = 1;
                        }

                    }
                    else
                    {

                        GridArray[mouseX, mouseY].Road = 1;
                        GridArray[mouseX, mouseY].Rotation = 0;
                    }


                }
            }





            // if the left mouse button is clicked a curve its placed, if the
cell picked is already a curve it will be rotated
            else if (Raylib.IsMouseButtonPressed(MouseButton.MOUSE_BUTTON_LEFT)
== true)
            {


                mouseX = Raylib.GetMouseX();
                mouseY = Raylib.GetMouseY();

                if (mouseX >= 262 && mouseX <= 1270 && mouseY <= 672)
                {

                    // we find the element of the array corrispondent to this
cordinates in our grid and we set to 1 to indicate that we drew a curve in that
cell
                    mouseX = (mouseX - 262) / cellxy;
                    mouseY = mouseY / cellxy;

                    if (GridArray[mouseX, mouseY].Road == 2)
                    {
                        if (GridArray[mouseX, mouseY].Rotation < 3)
                        {
                            GridArray[mouseX, mouseY].Rotation += 1;
                        }
                        else if (GridArray[mouseX, mouseY].Rotation == 3)
                        {
                            GridArray[mouseX, mouseY].Rotation = 0;
                        }
                    }
                    else
                    {

                        GridArray[mouseX, mouseY].Road = 2;
                        GridArray[mouseX, mouseY].Rotation = 0;
                    }


                }
            }

            // if the middle mouse button its clicked, the tile in the cell where
the mouse is when we press the button will be removed
            else if (Raylib.IsMouseButtonPressed(MouseButton.MOUSE_BUTTON_MIDDLE)
== true)
            {


                mouseX = Raylib.GetMouseX();
                mouseY = Raylib.GetMouseY();

                if (mouseX >= 262 && mouseX <= 1270 && mouseY <= 672)
                {

                    // we find the element of the array corrispondent to this
cordinates in our grid and we set to 1 to indicate that we drew a curve in that
cell
                    mouseX = (mouseX - 262) / cellxy;
                    mouseY = mouseY / cellxy;

                    GridArray[mouseX, mouseY].Road = 0;

                }
            }





        }

        // erase all the tiles in the grid
        public void Erase()
        {
            for (int i = 0; i < height; i++)
            {
                for (int z = 0; z < width; z++)
                {

                    GridArray[z,i].Road = 0;

                }

            }
        }
        // fills the empty part of the track with grass tiles
        public void FillTrack()
        {

            if (Raylib.IsKeyPressed(KeyboardKey.KEY_ENTER) == true)
            {
                Fill = true;
            }

        }

        // sets the first straight tile placed as the starting line for the track
        public void SetStartingPoint()
        {
            int y = 0;
            int x = 0;
            bool set = false;
            while (y<6 && set != true)
            {
                x = 0;
                while (x<9 && set != true)
                {
                    if (GridArray[x,y].Road == 1)
                    {
                        GridArray[x,y].StartingLine = true;
                        set = true;
                    }

                    x++;
                }

                y++;
            }

        }
        public void Update()
        {

            IsCellPicked();
            FillTrack();
        }
    }
}
```

3.4 TRACK.CS

```
using SharpDX.Direct3D11;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Raylib_CsLo;
using System.Threading.Tasks;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using System.Numerics;

namespace NEA
{
    [Serializable()]
    public class Track
    {

        private Grid trackGrid;
        private string trackName;
        private List<Waypoint> waypoints;


        public Grid TrackGrid { get => trackGrid; set => trackGrid = value; }
        public string TrackName { get => trackName; set => trackName = value; }
        public List<Waypoint> Waypoints { get => waypoints; set => waypoints =
value; }


        public Track(Grid grid, string name , List<Waypoint> waypoints)
        {
            TrackGrid = grid;
            TrackName = name;
            this.Waypoints = waypoints;



        }
        /// <summary>
        /// Generates the checkpoint utilised during the race to validate each
lap
        /// </summary>
        /// <returns></returns>
        public List<Checkpoint> GenerateCheckpoints()
        {
            List<Checkpoint> checkpoints = new List<Checkpoint>();
            Vector2 currWaip;
            Checkpoint temp;

            for (int i = 0; i < Waypoints.Count(); i++)
            {
                currWaip = new Vector2(Waypoints.ElementAt(i).x * 180 + 180,
Waypoints.ElementAt(i).y * 180);
                temp = new Checkpoint(new
Rectangle(currWaip.X,currWaip.Y,180,180));
                checkpoints.Add(temp);

            }

            return checkpoints;
        }
    }

}
```

3.5 VEHICLE.CS

```
using NEA.Screens;
using Raylib_CsLo;
using System;
using System.Collections.Generic;
using System.ComponentModel.Design;
using System.Linq;
using System.Numerics;
using System.Reflection.Metadata.Ecma335;
using System.Security.Cryptography;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Timers;

namespace NEA
{
    public class Vehicle
    {



        private Texture carTexture;
        private Vector2 position1 = new Vector2(1280 / 2, 720 / 2);
        private Vector2 gridPos = new Vector2();


        private Vector2 carVelocity = new Vector2(0, 0);
        private float carRotation = 90f;
        private float carAccel = 0.12f;
        private float maxSpeed = 3.8f;
        private float carRotSpeed = 2.7f;
        private float friction = 0.025f;
        private float carBrake = 0.03f;

        private float boost = 0;
        private float maxboost = 100;

        public float speed { get; set; }
        public Vector2 Position { get => Position1;}
        public Texture CarTexture { get => carTexture; set => carTexture = value;
}
        public Vector2 Position1 { get => position1; set => position1 = value; }
        public Vector2 GridPos { get => gridPos; set => gridPos = value; }
        public Vector2 CarVelocity { get => carVelocity; set => carVelocity =
value; }
        public float CarRotation { get => carRotation; set => carRotation =
value; }
        public float CarAccel { get => carAccel; set => carAccel = value; }
        public float MaxSpeed { get => maxSpeed; set => maxSpeed = value; }
        public float CarRotSpeed { get => carRotSpeed; set => carRotSpeed =
value; }
        public float Friction { get => friction; set => friction = value; }
        public float CarBrake { get => carBrake; set => carBrake = value; }
        public float Boost { get => boost; set => boost = value; }
        public float Maxboost { get => maxboost; set => maxboost = value; }


        public Vehicle(Vector2 startPosition, Texture texture)
        {

            CarTexture = texture;


        }

        public void Draw()
        {
            // divided by 4 since we scale down by 0.25
            Vector2 carCenter = new Vector2(CarTexture.width / 6,
CarTexture.height / 6);

            // Calculate the drawing position
            Vector2 drawCarPos = new Vector2(Position.X - carCenter.X, Position.Y
- carCenter.Y);

            // Draw the car using DrawTexturePro
            Raylib.DrawTexturePro(CarTexture, new Rectangle(0, 0,
CarTexture.width, CarTexture.height), new Rectangle(drawCarPos.X, drawCarPos.Y,
CarTexture.width/3, CarTexture.height/3), carCenter, CarRotation, Raylib.WHITE);



        }
        /// <summary>
        /// Movement system of each vehicle
        /// </summary>
        /// <param name="UP"></param>
        /// <param name="DOWN"></param>
        /// <param name="LEFT"></param>
        /// <param name="RIGHT"></param>
        /// <param name="BOOST"></param>
        public void Move(KeyboardKey UP,KeyboardKey DOWN,KeyboardKey LEFT,
KeyboardKey RIGHT, KeyboardKey BOOST)
        {



            if (Raylib.IsKeyDown(RIGHT))
            {
                CarRotation += CarRotSpeed;
                if (Boost <= Maxboost)
                {
                    Boost = Boost + 0.2f;
                }
            }
            if (Raylib.IsKeyDown(LEFT))
            {
                CarRotation -= CarRotSpeed;
                if (Boost <= Maxboost)
                {
                    Boost = Boost +  0.2f;
                }
            }
            if (Raylib.IsKeyDown(UP))
            {
                // Calculate acceleration vector based on car rotation
                float accelerationX = (float)Math.Sin(CarRotation * (Math.PI /
180.0)) * CarAccel;
                float accelerationY = (float)Math.Cos(CarRotation * (Math.PI /
180.0)) * CarAccel;

                // Update car velocity
                carVelocity.X += accelerationX;
                carVelocity.Y -= accelerationY;

                float speed = RayMath.Vector2Length(CarVelocity);
                //imposing the max soeed
                if (speed > MaxSpeed)
                {
                    CarVelocity = RayMath.Vector2Normalize(CarVelocity);
                    carVelocity.X *= MaxSpeed;
                    carVelocity.Y *= MaxSpeed;
                }



            }
            else if (Raylib.IsKeyDown(DOWN))
            {
                // Apply braking
                carVelocity.X *= 1.0f - CarBrake;
                carVelocity.Y *= 1.0f - CarBrake;
            }
            //use boost
            if (Raylib.IsKeyDown(BOOST) && Boost > 0)
            {

                float boostTimer;
                float boostLenght = 5;
                //increase maxspeed while using the boost
                MaxSpeed = 7f;


                if (Boost > 0)
                {
                        CarAccel = CarAccel + 3f;
                        Boost = Boost - 2f;
                }



            }
            else
            {

                while (CarAccel > 0.12f  || MaxSpeed > 3.7f)
                {
                    CarAccel -= 0.008f;
                    MaxSpeed -= 0.05f;
                }
                CarAccel = 0.12f;
                MaxSpeed = 3.7f;

            }



            // Apply friction
            carVelocity.X *= 1.0f - Friction;
            carVelocity.Y *= 1.0f - Friction;

            // Update car's position
            position1.X += CarVelocity.X;
            position1.Y += CarVelocity.Y;



        }










    }



}
```

3.6 PLAYERCAR.CS

```
using Raylib_CsLo;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Numerics;
using System.Text;
using System.Threading.Tasks;

namespace NEA
{
    public class PlayerCar : Vehicle
    {

        private KeyboardKey UP;
        private KeyboardKey DOWN;
        private KeyboardKey LEFT;
        private KeyboardKey RIGHT;
        private KeyboardKey BOOST;



        Vector2 carCenter;
        private Rectangle playerRect = new Rectangle();

        public Rectangle PlayerRect { get => playerRect; set => playerRect =
value; }

        public PlayerCar(KeyboardKey UP,KeyboardKey DOWN, KeyboardKey LEFT,
KeyboardKey RIGHT,KeyboardKey BOOST, Vector2 startPosition, Texture texture) :
base(startPosition, texture)
        {

            this.UP = UP;
            this.DOWN = DOWN;
            this.LEFT = LEFT;
            this.RIGHT = RIGHT;
            this.BOOST = BOOST;

            Position1 = new Vector2(startPosition.X,startPosition.Y);
            CarVelocity = new Vector2(0, 0);
            CarTexture = texture;
            carCenter = new Vector2(CarTexture.width / 6, CarTexture.height / 6);
            PlayerRect = new Rectangle(Position1.X, Position1.Y, texture.height /
3, texture.width / 3);


        }









        public void Update()
        {
            //calculates the rotation of the rectangle associated to the car
based on the car rotation, so we can be more accurate with collision by
            int moddedrotation = (int)CarRotation % 360;
            if (moddedrotation > 360)
            {
                moddedrotation = moddedrotation % 360;
            }
            if (moddedrotation < 45 || moddedrotation > 135 && moddedrotation <
225 || moddedrotation > 315)
            {
                PlayerRect = new Rectangle(Position1.X - carCenter.X -
CarTexture.width / 6, Position1.Y - carCenter.Y - CarTexture.height / 6,
CarTexture.width / 3, CarTexture.height / 3);

            }
            else
            {
                PlayerRect = new Rectangle(Position1.X - carCenter.X -
CarTexture.height / 6, Position1.Y - carCenter.Y - CarTexture.width / 6,
CarTexture.height / 3, CarTexture.width / 3);

            }

            Move(UP,DOWN,LEFT,RIGHT,BOOST);
        }

        public void Draw()
        {


            // Calculate the drawing position
            Vector2 drawCarPos = new Vector2(Position.X - carCenter.X, Position.Y
- carCenter.Y);

            // Draw the car using DrawTexturePro

            Raylib.DrawTexturePro(CarTexture, new Rectangle(0, 0,
CarTexture.width, CarTexture.height), new Rectangle(drawCarPos.X, drawCarPos.Y,
CarTexture.width / 3, CarTexture.height / 3), carCenter, CarRotation,
Raylib.WHITE);


        }








    }
}
```

3.7 CarAI.cs

```
using Raylib_CsLo;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Numerics;
using System.Text;
using System.Threading.Tasks;

namespace NEA
{
   public class CarAI
   {
        // List of waypoints that define the car's path
        private List<Waypoint> waypoints;
        //current waypoint index that the car is targeting
        private int currentWaypoint = 0;

        private Vector2 waypVector;
        Vector2 distVector;
        //car properties
        private Rectangle carRect;

        private float maxForce;
        private float maxSpeed;
        private Vector2 velocity;
        private Vector2 acceleration = Vector2.Zero;
        private Vector2 steer = Vector2.Zero;
        private Vector2 position = Vector2.Zero;
        private Vector2 carCenter;
        private Texture carTexture;
        private float carRotation;

        public Rectangle CarRect { get => carRect; set => carRect = value; }
        public float MaxForce { get => maxForce; set => maxForce = value; }
        public float MaxSpeed { get => maxSpeed; set => maxSpeed = value; }
        public Vector2 Velocity { get => velocity; set => velocity = value; }
        public Vector2 Acceleration { get => acceleration; set => acceleration =
value; }
        public Vector2 Steer { get => steer; set => steer = value; }
        public Vector2 Position { get => position; set => position = value; }
        public Vector2 CarCenter { get => carCenter; set => carCenter = value; }
        public Texture CarTexture { get => carTexture; set => carTexture = value;
}
        public float CarRotation { get => carRotation; set => carRotation =
value; }

        public CarAI(Vector2 startPosition, Texture texture , List<Waypoint>
waypoints) //: base(startPosition, texture)
        {
            position.X = startPosition.X * 180 + 270;
            position.Y = startPosition.Y * 180 + 90 + 10;
            CarTexture = texture;
            CarCenter = new Vector2(CarTexture.width / 6, CarTexture.height / 6);
            this.waypoints = waypoints;
            MaxSpeed = 4f;
            MaxForce = 3.0f;

            Velocity = Vector2.Zero;
            Acceleration = Vector2.Zero;
            CarRect = new Rectangle();

        }
        public void Draw()
        {

            CarRotation = (float)(Math.Atan2(Velocity.Y,
Velocity.X)/MathF.PI)*180 + 90;



            // Calculate the drawing position
            Vector2 drawCarPos = new Vector2(Position.X - CarCenter.X, Position.Y
- CarCenter.Y);

            // Draw the car using DrawTexturePro
            Raylib.DrawTexturePro(CarTexture, new Rectangle(0, 0,
CarTexture.width, CarTexture.height), new Rectangle(drawCarPos.X, drawCarPos.Y,
CarTexture.width / 3, CarTexture.height / 3), CarCenter, CarRotation,
Raylib.WHITE);



        }

        /// <summary>
        /// Movement system for the cpu car
        /// </summary>
        public void Move()
        {

            //calcualtes next waypoint position
            int nextWaypointX = waypoints.ElementAt((currentWaypoint + 1) %
waypoints.Count ).x * 180 + 180+ 90;
            int nextWaypointY = waypoints.ElementAt((currentWaypoint + 1) %
waypoints.Count).y * 180 + 90;


            Rectangle temp = new Rectangle(nextWaypointX, nextWaypointY, 180,
180);

            CarRect = new (Position.X, Position.Y, CarTexture.width / 3,
CarTexture.height / 3);

            waypVector = new Vector2(nextWaypointX,nextWaypointY);
            Seek(waypVector);

            if (Raylib.CheckCollisionCircleRec(new Vector2 (nextWaypointX,
nextWaypointY),10 , CarRect) == true)
            {
                currentWaypoint = (currentWaypoint + 1)% waypoints.Count;
            }


            //NextVector();


        }

        public void ApplyForce(Vector2 force)
        {
            Acceleration += force;
        }
        /// <summary>
        /// switches to the vector guiding to the next checkpoint
        /// </summary>
        public void NextVector()
        {

            distVector = new Vector2((waypoints.ElementAt((currentWaypoint + 1) %
waypoints.Count).x * 180 + 90 + 180) - Position.X,
(waypoints.ElementAt((currentWaypoint + 1) % waypoints.Count).y * 180 + 90) -
Position.Y);
        }
        /// <summary>
        /// calculate the vector that the cpu car will do to reach the next
waypoint
        /// </summary>
        /// <param name="target"></param>
        public void Seek(Vector2 target)
        {
            Vector2 desired = Vector2.Subtract(target,Position);
            desired = Vector2.Normalize(desired) * MaxSpeed;
            Steer = Vector2.Subtract(desired, Velocity);
            Steer = Vector2.Clamp(Steer, Vector2.Negate(new Vector2(MaxForce,
MaxForce)), new Vector2(MaxForce, MaxForce));
            ApplyForce(Steer);
        }

        public void Update()
        {
             Move();
             Velocity += Acceleration;
             Velocity = Vector2.Clamp(Velocity,Vector2.Negate(new
Vector2(MaxSpeed,MaxSpeed)), new Vector2(MaxSpeed, MaxSpeed));
             Position += Velocity;
             Acceleration *= 0;
            //calculates the rotation of the rectangle associated to the car
based on the car rotation, so we can be more accurate with collision
            int moddedrotation = (int)CarRotation % 360;
            if (moddedrotation > 360)
            {
                moddedrotation = moddedrotation % 360;
            }
            if (moddedrotation < 45 || moddedrotation > 135 && moddedrotation <
225 || moddedrotation > 315 )
            {
                CarRect = new Rectangle(Position.X - CarCenter.X -
CarTexture.width / 6, Position.Y - CarCenter.Y - CarTexture.height / 6,
CarTexture.width / 3, CarTexture.height / 3);

            }
            else
            {
                CarRect = new Rectangle(Position.X - CarCenter.X -
CarTexture.height / 6, Position.Y - CarCenter.Y - CarTexture.width / 6,
CarTexture.height / 3, CarTexture.width / 3);

            }




        }
    }
}
```

3.8 STAR.CS

```
using Raylib_CsLo;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Numerics;
using System.Text;
using System.Threading.Tasks;

namespace NEA.Starfield
{
    public class Star
    {
        private float x;
        private float y;
        private float z;

        private float previousZ;

        public Star()
        {
            // 3D starting position of the star

            x = Raylib.GetRandomValue(-Program.screenwidth, Program.screenwidth);
            y = Raylib.GetRandomValue(-Program.screenheight, 
Program.screenheight);
            z = Raylib.GetRandomValue(0,Program.screenwidth);
            previousZ = z;
        }
        public void Update()
        {
            z = z - StarField.Speed;

            //If the star goes beyond the screen and so z is less than 1 it 
resets its position
            if (z < 1)
            {
                x = Raylib.GetRandomValue(-Program.screenwidth, 
Program.screenwidth);
                y = Raylib.GetRandomValue(-Program.screenheight, 
Program.screenheight);
                z = Program.screenwidth;
                previousZ = z;
            }
        }
        public void Draw()
        {

            // Calculate the screen position of the star by mapping the values
            int sx = (int)Map(x/z,0,1,0, Program.screenwidth);
            int sy = (int)Map(y/z, 0, 1, 0, Program.screenheight);

            float size = Map(z, 0, Program.screenwidth, 14, 0);

            // Calculate the previous screen position of the star for drawing the 
trail effect
            int px = (int)Map(x / previousZ, 0, 1, 0, Program.screenwidth);
            int py = (int)Map(y / previousZ, 0, 1, 0, Program.screenheight);

            previousZ = z;
            // Generate a random color for the star
            Random random = new Random();
            int r = random.Next(0,255);
            int g = random.Next(0,255);
            int b = random.Next(0,255);
            int a = random.Next(0,255);
            Color color = new Color(r,g,b,a);
            Raylib.DrawLineEx(new Vector2( (int)px + Program.screenwidth / 2, 
(int)py + Program.screenheight / 2) , new Vector2((int)sx + Program.screenwidth / 
2, (int)sy + Program.screenheight / 2),3.5f, color);
        }

        public static float Map(float value, float low1, float high1,float low2, 
float high2)
        {
            //normalize value to be between 0 and  1
            float normalizedValue = (value - low1) / (high1 - low1);
            //now we map the value to the new range from 0 to 1
            float mappedValue = low2 + (normalizedValue * (high2 - low2));

            return mappedValue;
        }
    }
}
```

3.9 STARFIELD.CS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Raylib_CsLo;

namespace NEA.Starfield
{

    public class StarField
    {
        private static float speed;
        //array of stars that compone the starfield
        private Star[] stars = new Star[1000];

        public static float Speed { get => speed; set => speed = value; }

        public StarField()
        {
            Setup();
        }
        public void Setup()
        {
            //initialize all stars in the array
            for (int i = 0; i < stars.Length; i++)
            {
                 stars[i] = new Star();


            }
        }
        //draw starfield
        public void Draw()
        {
            // update star movement speed according to mouse x position
            Speed = Star.Map(Raylib.GetMouseX(),0,Program.screenwidth,0,55);
            for (int i = 0; i < stars.Length; i++)
            {

                stars[i].Update();

                stars[i].Draw();
            }
        }
    }


}
```

3.10 Checkpoint

```
using System;
using System.Collections.Generic;

using System.Linq;
using System.Numerics;
using System.Text;
using System.Threading.Tasks;
using Raylib_CsLo;

namespace NEA
{
    [Serializable]
    public class Checkpoint
    {
        //rectangle for collision area of checkpoint
        private Rectangle checkRect = new Rectangle();


        public Checkpoint(Rectangle checkRect)
        {
            this.CheckRect = checkRect;

        }

        public Rectangle CheckRect { get => checkRect; set => checkRect = value;
}
    }
}
```

3.11 CheckpointManager.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NEA
{
    public class CheckpointManager
    {
        private Queue<Checkpoint> p1Queue = new Queue<Checkpoint>();
        private Queue<Checkpoint> p2Queue = new Queue<Checkpoint>();
        private List<Checkpoint> checkpList;

        private int p1Lapsleft;
        private int p2Lapsleft;

        public Queue<Checkpoint> P2Queue { get => p2Queue; set => p2Queue =
value; }
        public Queue<Checkpoint> P1Queue { get => p1Queue; set => p1Queue =
value; }
```

```
        public List<Checkpoint> CheckpList { get => checkpList; set => checkpList
= value; }
        public int P1Lapsleft { get => p1Lapsleft; set => p1Lapsleft = value; }
        public int P2Lapsleft { get => p2Lapsleft; set => p2Lapsleft = value; }


        public CheckpointManager(List<Checkpoint> checkps, int lapsNumber)
        {
            P1Lapsleft = lapsNumber;
            P2Lapsleft = lapsNumber;
            CheckpList = checkps;
            for (int i = 0; i < CheckpList.Count(); i++)
            {
                P1Queue.Enqueue(CheckpList.ElementAt(i));
                P2Queue.Enqueue(CheckpList.ElementAt(i));
            }
            // queue another time the first cell since has to be trigerred at the
start and when we arrive
            P1Queue.Enqueue(CheckpList.ElementAt(0));
            P2Queue.Enqueue(CheckpList.ElementAt(0));

        }

        /// <summary>
        /// Check that when the first player crosses the starting line, all the
checkpoints have been passed and in case decreases the laps left
        /// </summary>
        /// <returns></returns>
        public bool p1IsLapValid()
        {
            bool IsValid = false;
            if (P1Queue.Count == 0)
            {
                IsValid = true;
                P1Lapsleft--;
                for (int i = 0; i < CheckpList.Count(); i++)
                {
                    P1Queue.Enqueue(CheckpList.ElementAt(i));
                }

            }

            return IsValid;
        }

        /// <summary>
        /// Check that when the second player crosses the starting line, all the
checkpoints have been passed and in case decreases the laps left
        /// </summary>
        /// <returns></returns>
        public bool p2IsLapValid()
        {
            bool IsValid = false;
            if (P2Queue.Count == 0)
            {
                P2Lapsleft--;
                IsValid = true;
                for (int i = 0; i < CheckpList.Count(); i++)
                {
                    P2Queue.Enqueue(CheckpList.ElementAt(i));
                }

            }
            return IsValid;
        }


        /// <summary>
        /// deques every checkpoint when its passed by the first player
        /// </summary>
        public void p1CheckpointPassed()
        {

            P1Queue.Dequeue();



        }
        /// <summary>
        /// deques every checkpoint when its passed by the second player
        /// </summary>
        public void p2CheckpointPassed()
        {

            P2Queue.Dequeue();


        }

    }
}
```

3.12 Track.cs

```
using SharpDX.Direct3D11;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Raylib_CsLo;
using System.Threading.Tasks;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using System.Numerics;

namespace NEA
{
    [Serializable()]
    public class Track
    {

        private Grid trackGrid;
        private string trackName;
        private List<Waypoint> waypoints;


        public Grid TrackGrid { get => trackGrid; set => trackGrid = value; }
        public string TrackName { get => trackName; set => trackName = value; }
        public List<Waypoint> Waypoints { get => waypoints; set => waypoints =
value; }


        public Track(Grid grid, string name , List<Waypoint> waypoints)
        {
            TrackGrid = grid;
            TrackName = name;
            this.Waypoints = waypoints;



        }
        /// <summary>
        /// Generates the checkpoint utilised during the race to validate each
lap
        /// </summary>
        /// <returns></returns>
        public List<Checkpoint> GenerateCheckpoints()
        {
            List<Checkpoint> checkpoints = new List<Checkpoint>();
            Vector2 currWaip;
            Checkpoint temp;

            for (int i = 0; i < Waypoints.Count(); i++)
            {
                currWaip = new Vector2(Waypoints.ElementAt(i).x * 180 + 180,
Waypoints.ElementAt(i).y * 180);
                temp = new Checkpoint(new
Rectangle(currWaip.X,currWaip.Y,180,180));
                checkpoints.Add(temp);

            }

            return checkpoints;
        }
    }

}
```

3.13 TrackManager.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Reflection.Metadata;
using System.Security.Cryptography.X509Certificates;
using System.Text;
using System.Threading.Tasks;

namespace NEA
{
    public static class TrackManager
    {
        public const string trackFileName = "Tracks.bin";

        /// <summary>
        /// Save tracks to a binary file
        /// </summary>
        /// <param name="trackList">tracklist loaded intO FILE</param>
        public static List<Track> SaveTracks(List<Track> trackList)
        {


            try
            {
                using (Stream stream = File.Open(trackFileName, FileMode.Create))
                {
                    var bformatter = new
System.Runtime.Serialization.Formatters.Binary.BinaryFormatter();
                    bformatter.Serialize(stream, trackList);


                }
            }
            catch (Exception e)
            {
                Console.WriteLine("Error saving tracks " + e.Message);

            }

            return trackList;


        }
        /// <summary>
        /// //Access the binary file, reads it and associate the list of tracks
stored to the variavle trackList
        /// </summary>
        /// <param name="trackList"></param>
        public static List<Track> LoadTracks(List<Track> trackList)
        {
            try
            {
                using (Stream stream = File.Open(trackFileName, FileMode.Open))
                {
                    var bformatter = new
System.Runtime.Serialization.Formatters.Binary.BinaryFormatter();

                    trackList = (List<Track>)bformatter.Deserialize(stream);
                }


            }
            catch (Exception e)
            {
                Console.WriteLine("Error loading tracks " + e.Message);

            }


            return trackList;
        }
    }
}
```

3.14 Waypoint.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Numerics;
using System.Text;
using System.Threading.Tasks;

namespace NEA
{
    [Serializable]
    public class Waypoint
    {
        public int x;
        public int y;


        public Waypoint()
        {
        }
    }
}
```

3.15 SplashScreen

```
using NEA.Starfield;
using Raylib_CsLo;

using System;
using System.Collections.Generic;
using System.Linq;
using System.Numerics;
using System.Text;
using System.Threading.Tasks;

namespace NEA.Screens
{
    public class SplashScreen : IGameScreen
    {
        private StarField starField = new StarField();

        private IGameScreenManager m_screenManager;
        public SplashScreen(IGameScreenManager gameScreenManager)
        {
            m_screenManager = gameScreenManager;
        }

        public void Dispose()
        {

        }

        public void Draw()
        {
            Raylib.ClearBackground(Raylib.BLACK);
            //starfield animation
            starField.Draw();

            if (RayGui.GuiButton(new Rectangle(Program.screenwidth - 300,0, 300,
100), "PLAY") == true)
            {
                m_screenManager.PushScreen(new
TrackSelectionScreen(m_screenManager));
            }
            //TITLE TEXTURE
            Raylib.DrawTexturePro(Program.CARZ,new
Rectangle(0,0,Program.CARZ.width,Program.CARZ.height), new Rectangle(1280/2 -145,
720/2 - 100, Program.CARZ.width *4, Program.CARZ.height*4),new
Vector2(0,0),0,Raylib.WHITE);
        }

        public void HandleInput()
        {

        }

        public void Update()
        {

        }
    }
}
```

3.16 TrackSelectionScreen.cs

```
using Raylib_CsLo;
using SharpDX.Direct3D11;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Numerics;
using System.Text;
using System.Threading.Tasks;
using System.IO;



namespace NEA.Screens
{
    public class TrackSelectionScreen : IGameScreen
    {
        private IGameScreenManager m_screenManager;
        public static Grid grid = new Grid();
        private List<Track> trackList = new List<Track>();
        public static int currentTrack = 0;

        public List<Track> TrackList { get => trackList; set => trackList =
value; }


        public TrackSelectionScreen(IGameScreenManager gameScreenManager)
        {
            m_screenManager = gameScreenManager;
            //reading list of tracks
            TrackList = TrackManager.LoadTracks(TrackList);

        }

        public void Dispose()
        {

        }

        public void Draw()
        {
            try
            {
                //show different track on screen
                if (File.Exists(TrackManager.trackFileName))
                {
                    if (TrackList.Count != 0)
                    {
                        Raylib.DrawText("TRACK: " +
TrackList.ElementAt(currentTrack).TrackName.ToUpper(), 725, 220, 20,
Raylib.LIME);
                        DrawTrackPreview();
                    }

                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);

            }




        }

        /// <summary>
        /// Handles the input of the button pressed
        /// </summary>
        public void HandleInput()
        {
            if (RayGui.GuiButton(new Rectangle(75, 100, 200, 75), "SELECT TRACK")
== true)
            {
                if (TrackList.Count > 0)
                {
                    Vector2 carPosition = CarStartingGridPos();
                    m_screenManager.PushScreen(new
CarSelectionScreen(m_screenManager, TrackList[currentTrack],carPosition));
                }
                else
                {
                    m_screenManager.PushScreen(new
TrackEditorScreen(m_screenManager,TrackList));
                }

            }
            if (RayGui.GuiButton(new Rectangle(75, 250, 200, 75), "TRACK EDITOR")
== true)
            {
                m_screenManager.PushScreen(new
TrackEditorScreen(m_screenManager,TrackList));
            }

            if (RayGui.GuiButton(new Rectangle(75, 400, 200, 75), "DELETE TRACK")
== true)
            {
                try
                {
                    if (TrackList.Count > 0)
                    {
                        TrackList.RemoveAt(currentTrack);
                        if (currentTrack == 0)
                        {

                        }
                        else
                        {
                            currentTrack--;
                        }
                        TrackManager.SaveTracks(TrackList);
                    }
                }
                catch (Exception e)
                {

                    Console.WriteLine(e.Message);
                }




            }
            //buttons to go through the the different tracks

            if (RayGui.GuiButton(new Rectangle(585,300,25,25),"<"))
            {
                if (currentTrack != 0)
                {
                    currentTrack --;
                }
                else if (TrackList.Count != 0)
                {

                    currentTrack = TrackList.Count -1;
                }
            }

            if (RayGui.GuiButton(new Rectangle(1100, 300, 25, 25), ">"))
            {
                if (currentTrack != TrackList.Count - 1)
                {
                    currentTrack ++;
                }

                else
                {
                    currentTrack =0;
                }
            }
        }



        /// <summary>
        /// draws the track in smaller size to give the user a preview of the
track before selecting it
        /// </summary>
        public void DrawTrackPreview()
        {


            Cell[,] trackGrid =
TrackList.ElementAt(currentTrack).TrackGrid.GridArray;


            Vector2 roadCenter = new Vector2(Grid.roadTexture.width / 2,
Grid.roadTexture.height / 2);

            int cellx = 0;
            int celly = 0;
            //We draw the preview of the tracks in a scaled grid, using 
Raylib.DrawtexturePro to scale it to the 25%

            for (int i = 0; i < Grid.height; i++)
            {
                for (int z = 0; z < Grid.width; z++)
                {

                    if (trackGrid[z, i].Road == 0)
                    {
                        Raylib.DrawTexturePro(Grid.grassTexture, new Rectangle(0, 
0, Grid.roadTexture.width, Grid.roadTexture.height), new Rectangle((z * (112 / 
4)) + roadCenter.X + ((500 + 1100) / 2) - Grid.roadTexture.width, i * (112 / 4) + 
roadCenter.X + (Program.screenheight / 2 - Grid.roadTexture.height * (18/10)) - 
50, Grid.roadTexture.width / 4, Grid.roadTexture.height / 4), roadCenter/4, 
trackGrid[z, i].Rotation * 90, Raylib.WHITE);

                    }

                    if (trackGrid[z, i].Road == 1)
                    {


                        Raylib.DrawTexturePro(Grid.roadTexture, new Rectangle(0, 
0, Grid.roadTexture.width , Grid.roadTexture.height ), new Rectangle((z * (112 / 
4)) + roadCenter.X + ((500 + 1100) / 2) - Grid.roadTexture.width, i * (112 / 4) + 
roadCenter.X + (Program.screenheight / 2 - Grid.roadTexture.height * (18/10)) - 
50, Grid.roadTexture.width / 4, Grid.roadTexture.height / 4), roadCenter/4, 
trackGrid[z, i].Rotation * 90 , Raylib.WHITE);

                    }
                    else if (trackGrid[z, i].Road == 2)
                    {
                        Raylib.DrawTexturePro(Grid.curveTexture, new Rectangle(0, 
0, Grid.curveTexture.width , Grid.curveTexture.height ), new Rectangle((z * (112 
/ 4)) + roadCenter.X + ((500 + 1100) / 2) - Grid.roadTexture.width, i * (112 / 4) 
+ roadCenter.X + (Program.screenheight / 2 - Grid.roadTexture.height * (18 / 10)) 
- 50, Grid.curveTexture.width / 4, Grid.curveTexture.height / 4), roadCenter / 4, 
trackGrid[z, i].Rotation * 90, Raylib.WHITE);
                    }


                }




            }

        }


        /// <summary>
        /// Calculate the starting line of the track and returns the cell 
position in the grid
        /// </summary>
        /// <returns></returns>
        public Vector2 CarStartingGridPos()
        {
            bool found = false;
            Cell[,] trackGrid =
TrackList.ElementAt(currentTrack).TrackGrid.GridArray;
            int x = 0;
            int y = 0;
            Vector2 carPos = Vector2.Zero;

            while (y < 6 && found != true)
            {
                x = 0;
                while (x < 9 && found != true)
                {
                    if (trackGrid[x, y].StartingLine == true)
                    {
                        found = true;
                        carPos = new Vector2(x, y);
                    }






                     x++;
                }

                y++;
            }

            return carPos;
        }

        public void Update()
        {

        }
    }
}
```

3.17 TrackEditorScreen.cs

```
using NEA.Screens;
using Raylib_CsLo;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Numerics;
using System.Text;
using System.Threading.Tasks;

namespace NEA
{
    public class TrackEditorScreen:IGameScreen
    {
        private IGameScreenManager m_gameScreen;
        private Grid grid = new Grid();
        private bool editMode;
        public string trackName = "";
        private bool errorMessage = false;
        private List<Waypoint> waypoints = new List<Waypoint>();
        private List<Checkpoint> checkpoints = new List<Checkpoint>();
        private List<Track> trackList = new List<Track>();
        public bool ErrorMessage { get => errorMessage; set => errorMessage = 
value; }
        public List<Waypoint> Waypoints { get => waypoints; set => waypoints = 
value; }
        public List<Checkpoint> Checkpoints { get => checkpoints; set => 
checkpoints = value; }
        public List<Track> TrackList { get => trackList; set => trackList = 
value; }
        public Grid Grid { get => grid; set => grid = value; }

        public TrackEditorScreen(IGameScreenManager gameScreen, List<Track> 
tracks)
        {
            m_gameScreen = gameScreen;
            TrackList = tracks;

        }

        public void Dispose()
        {

        }

        public void Draw()
        {


            Gui.GuiTextInputBox(new Rectangle(25, 40, 200, 100), "","","",ref 
trackName);

            Cordinate cordinate;



           if (RayGui.GuiButton(new Rectangle(205,40,22,22),"X") == true)
           {
                m_gameScreen.PopScreen();
           }
           if (RayGui.GuiButton(new Rectangle(25, 100, 100, 35),"ERASE") == true)
           {
                Grid.Erase();
           }
           else if(RayGui.GuiButton(new Rectangle(125, 100, 100, 35),"SAVE") == 
true)
           {
                cordinate = firstStraightCell();
                if (cordinate.x != -1 && cordinate.y != -1)
                {
                    Traverse(cordinate);



                    if (IsTrackValid() == true && IsTrackOnly() == true)
                    {


                        Grid.Fill = true;
                        Grid.SetStartingPoint();
                        Track newTrack = new Track(Grid, trackName, Waypoints);
                        TrackList.Add(newTrack);
                        TrackManager.SaveTracks(TrackList);
                        m_gameScreen.PopScreen();
                        TrackSelectionScreen.currentTrack++;
                    }
                    else
                    {
                        ErrorMessage = true;

                    }
                }
                else
                {

                    ErrorMessage = true;
                }



            }

             if (ErrorMessage == true)
             {
                 bool editmode = false;

                 if (RayGui.GuiTextBox(new Rectangle(25,200 ,200, 100), "THE TRACK
IS NOT VALID",25,editmode))
                 {
                     Grid.Erase();
                     ErrorMessage = false;
                 }


             }


            Grid.Draw(112,262,1, 150);

        }

        //Save track on the text file

        public void HandleInput()
        {

        }


        /// <summary>
        /// Checks if track created by the user is valid by checking that the
road tiles use are compatible and dont overlap
        /// </summary>
        /// <returns></returns>

    public bool IsTrackValid()
    {

             int validNeighb = 0;
             bool validCell = true;
             bool validLine = false;
             Cell upNeighb;
             Cell bottNeighb;
             Cell rightNeighb;
             Cell leftNeighb;
             Cell currNeighb;
             int i = 0;
             int z = 0;
            while (i < Grid.height && validCell != false )
            {
                validLine = false;

                while (z < Grid.width)
                {

                    currNeighb = Grid.GridArray[z, i];
                    upNeighb = new Cell();
                    bottNeighb = new Cell();
                    rightNeighb = new Cell();
                    leftNeighb = new Cell();
                    validNeighb = 0;
                   //straight works with current neighb


                    if (z != 0)
                    {
                        leftNeighb = Grid.GridArray[z - 1, i];
                    }
                    if (z != 8)
                    {
                        rightNeighb = Grid.GridArray[z + 1, i];
                    }
                    if (i != 0)
                    {
                        upNeighb = Grid.GridArray[z, i - 1];
                    }
                    if (i != 5)
                    {
                        bottNeighb = Grid.GridArray[z, i + 1];
                    }


                    if (currNeighb.Road != 0)
                    {
                        validLine = true;
                        if (currNeighb.Road == 1)
                        {



                            if ((leftNeighb.Road == 1 && leftNeighb.Rotation ==
0) || (leftNeighb.Road == 2 && currNeighb.Rotation == 0 && (leftNeighb.Rotation
== 3 || leftNeighb.Rotation == 2)))
                            {

                                validNeighb++;
                            }
                            if ((rightNeighb.Road == 1 && rightNeighb.Rotation ==
0) || (rightNeighb.Road == 2 && currNeighb.Rotation == 0 && (rightNeighb.Rotation
== 0 || rightNeighb.Rotation == 1)))
                            {
                                validNeighb++;
                            }
                            if ((upNeighb.Road == 1 && upNeighb.Rotation == 1) ||
(upNeighb.Road == 2 && currNeighb.Rotation == 1 && (upNeighb.Rotation == 0 ||
upNeighb.Rotation == 3)))
                            {
                                validNeighb++;
                            }
                            if ((bottNeighb.Road == 1 && bottNeighb.Rotation ==
1) || (bottNeighb.Road == 2 && currNeighb.Rotation == 1 && (bottNeighb.Rotation
== 1 || bottNeighb.Rotation == 2)))
                            {
                                validNeighb++;
                            }


                        }
                        else if (currNeighb.Road == 2)
                        {
                            if (currNeighb.Rotation == 0)
                            {

                                if ((bottNeighb.Road == 1 && bottNeighb.Rotation
== 1) || (bottNeighb.Road == 2 && (bottNeighb.Rotation == 1 ||
bottNeighb.Rotation == 2)))
                                {
                                    validNeighb++;
                                }
                                if ((leftNeighb.Road == 1 && leftNeighb.Rotation
== 0) || (leftNeighb.Road == 2 && (leftNeighb.Rotation == 2 ||
leftNeighb.Rotation == 3)))
                                {
                                    validNeighb++;
                                }


                            }
                            if (currNeighb.Rotation == 1)
                            {

                                if ((leftNeighb.Road == 1 && leftNeighb.Rotation
== 0) || (leftNeighb.Road == 2 && (leftNeighb.Rotation == 2 ||
leftNeighb.Rotation == 3)))
                                {
                                    validNeighb++;
                                }
                                if ((upNeighb.Road == 1 && upNeighb.Rotation ==
1) || (upNeighb.Road == 2 && (upNeighb.Rotation == 0 || upNeighb.Rotation == 3)))
                                {
                                    validNeighb++;
                                }

                            }
                            if (currNeighb.Rotation == 2)
                            {

                                if ((upNeighb.Road == 1 && upNeighb.Rotation ==
1) || (upNeighb.Road == 2 && (upNeighb.Rotation == 0 || upNeighb.Rotation == 3)))
                                {
                                    validNeighb++;
                                }
                                if ((rightNeighb.Road == 1 &&
rightNeighb.Rotation == 0) || (rightNeighb.Road == 2 && (rightNeighb.Rotation ==
0 || rightNeighb.Rotation == 1)))
                                {
                                    validNeighb++;
                                }

                            }
                            if (currNeighb.Rotation == 3)
                            {
                                if ((bottNeighb.Road == 1 && bottNeighb.Rotation
== 1) || (bottNeighb.Road == 2 && (bottNeighb.Rotation == 1 ||
bottNeighb.Rotation == 2)))
                                {
                                    validNeighb++;
                                }
                                if ((rightNeighb.Road == 1 &&
rightNeighb.Rotation == 0) || (rightNeighb.Road == 2 && (rightNeighb.Rotation ==
0 || rightNeighb.Rotation == 1)))
                                {
                                    validNeighb++;
                                }

                            }
                        }

                        if (validNeighb != 2 && validLine == true)
                        {

                            validCell = false;
                        }



                    }

                    z++;
                }
                z = 0;
                i++;
            }

            return validCell;

        }

        //finds the first cell from where we start to traverse the graph
        public Cordinate firstStraightCell()
        {
            Cordinate cordinate;
            cordinate.x = -1;
            cordinate.y = -1;
            int x = 0;
            int y = 0;
            bool found = false;
            while (y < 6 && found == false)
            {
                x = 0;

                while (x < 9 && found == false)
                {
                    if (Grid.GridArray[x, y].Road == 1)
                    {

                        cordinate.x = x;
                        cordinate.y = y;
                        found = true;
                    }

                    x++;
                }
                y++;
            }

            return cordinate;
        }
        /// <summary>
        /// Traverse the track recursevely as a graph and put as traversed the
traversed cells
        /// </summary>
        /// <param name="cordinate"></param>
        public void Traverse(Cordinate cordinate)
        {
            bool trackTrav = false;

            Cell currCell = Grid.GridArray[cordinate.x, cordinate.y];
            Cell upNeighb = null;
            Cell bottNeighb = null;
            Cell rightNeighb = null;
            Cell leftNeighb = null;
            Waypoint currWaipoint = new Waypoint();

            currWaipoint.x = cordinate.x;
            currWaipoint.y = cordinate.y;




            Waypoints.Add(currWaipoint);



                if (currCell.Road == 1 && currCell.Rotation == 0)
                {

                    if (cordinate.x > 0)
                    {
                        leftNeighb = Grid.GridArray[cordinate.x - 1,
cordinate.y];
                    }
                    if (cordinate.x < 8)
                    {
                        rightNeighb = Grid.GridArray[cordinate.x + 1,
cordinate.y];
                    }



                    currCell.Traversed = true;

                    if (rightNeighb != null == rightNeighb.Traversed == false)
                    {

                        Traverse(new Cordinate(cordinate.x + 1, cordinate.y));
                    }
                    else if (leftNeighb != null && leftNeighb.Traversed == false)
                    {

                        Traverse(new Cordinate(cordinate.x - 1, cordinate.y));
                    }
                    else
                    {
                        trackTrav = true;
                    }
                }

                if (currCell.Road == 1 && currCell.Rotation == 1)
                {
                    if (cordinate.y > 0)
                    {
                        upNeighb = Grid.GridArray[cordinate.x, cordinate.y - 1];
                    }
                    if (cordinate.y < 5)
                    {
                        bottNeighb = Grid.GridArray[cordinate.x, cordinate.y +
1];
                    }


                    currCell.Traversed = true;
                    if (upNeighb != null && upNeighb.Traversed == false)
                    {
                        Traverse(new Cordinate(cordinate.x, cordinate.y - 1));
                    }
                    else if (bottNeighb != null && bottNeighb.Traversed == false)
                    {
                        Traverse(new Cordinate(cordinate.x, cordinate.y + 1));
                    }
                    else
                    {
                        trackTrav = true;
                    }
                }

                if (currCell.Road == 2 && currCell.Rotation == 0)
                {

                    if (cordinate.x > 0)
                    {
                        leftNeighb = Grid.GridArray[cordinate.x - 1,
cordinate.y];
                    }
                    if (cordinate.y < 5)
                    {
                        bottNeighb = Grid.GridArray[cordinate.x, cordinate.y +
1];
                    }

                    currCell.Traversed = true;
                    if (leftNeighb != null && leftNeighb.Traversed == false)
                    {
                        Traverse(new Cordinate(cordinate.x - 1, cordinate.y));
                    }
                    else if (bottNeighb != null && bottNeighb.Traversed == false)
                    {
                        Traverse(new Cordinate(cordinate.x, cordinate.y + 1));
                    }
                    else
                    {
                        trackTrav = true;
                    }
                }

                if (currCell.Road == 2 && currCell.Rotation == 1)
                {
                    if (cordinate.x > 0)
                    {
                        leftNeighb = Grid.GridArray[cordinate.x - 1,
cordinate.y];
                    }
                    if (cordinate.y > 0)
                    {
                        upNeighb = Grid.GridArray[cordinate.x, cordinate.y - 1];
                    }

                    currCell.Traversed = true;
                    if (leftNeighb != null && leftNeighb.Traversed == false)
                    {
                        Traverse(new Cordinate(cordinate.x - 1, cordinate.y));
                    }
                    else if (upNeighb != null && upNeighb.Traversed == false)
                    {
                        Traverse(new Cordinate(cordinate.x, cordinate.y - 1));
                    }
                    else
                    {
                        trackTrav = true;
                    }
                }

                if (currCell.Road == 2 && currCell.Rotation == 2)
                {
                    if (cordinate.x < 8)
                    {
                        rightNeighb = Grid.GridArray[cordinate.x + 1,
cordinate.y];
                    }

                    if (cordinate.y > 0)
                    {
                        upNeighb = Grid.GridArray[cordinate.x, cordinate.y - 1];
                    }
                    currCell.Traversed = true;
                    if (rightNeighb != null && rightNeighb.Traversed == false)
                    {
                        Traverse(new Cordinate(cordinate.x + 1, cordinate.y));
                    }
                    else if (upNeighb != null && upNeighb.Traversed == false)
                    {
                        Traverse(new Cordinate(cordinate.x, cordinate.y - 1));
                    }
                    else
                    {
                        trackTrav = true;
                    }
                }

                if (currCell.Road == 2 && currCell.Rotation == 3)
                {
                    if (cordinate.x < 8)
                    {
                        rightNeighb = Grid.GridArray[cordinate.x + 1,
cordinate.y];
                    }

                    if (cordinate.y < 5)
                    {
                        bottNeighb = Grid.GridArray[cordinate.x, cordinate.y +
1];
```

```
                }
                currCell.Traversed = true;
                if (rightNeighb != null && rightNeighb.Traversed == false)
                {
                    Traverse(new Cordinate(cordinate.x + 1, cordinate.y));
                }
                else if (bottNeighb != null && bottNeighb.Traversed == false)
                {
                    Traverse(new Cordinate(cordinate.x, cordinate.y + 1));
                }
                else
                {
                    trackTrav = true;
                }

            }


        }
        /// <summary>
        /// checks that the user created more than 1 valid track at the same time
by going throught the array and checking all cell have been traversed
        /// </summary>
        /// <returns></returns>
        public bool IsTrackOnly()
        {
            int x = 0;
            int y = 0;
            bool valid = true;


            while (y < 6 && valid != false)
            {

                x = 0;
                while (x < 9)
                {
                    if (Grid.GridArray[x,y].Road != 0)
                    {

                        if (Grid.GridArray[x, y].Traversed == false)
                        {
                            valid = false;
                        }

                    }

                    x++;
                }
                y++;
            }

            return valid;
        }

        public void Update()
        {
            Grid.Update();
        }


        public struct Cordinate
        {
            public int x;
            public int y;

            public Cordinate(int x, int y)
            {
                this.x = x;
                this.y = y;
            }
        }

    }
}
```

3.18 CarSelectionScreen

```
using Raylib_CsLo;

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Numerics;
using System.Text;
using System.Threading.Tasks;

namespace NEA.Screens
{
    public class CarSelectionScreen : IGameScreen
    {
        private IGameScreenManager m_gameScreen;
        public List<PlayerCar> p1_vehicles = new List<PlayerCar>();
        public List<PlayerCar> p2_vehicles = new List<PlayerCar>();
        bool playCPU = false;
        private int p1currentVehicle = 0;
        private int p2currentVehicle = 0;
        private Track track;
        private PlayerCar p1vehicle1;
        private PlayerCar p1vehicle2;
        private PlayerCar p1vehicle3;

        private PlayerCar p2vehicle1;
        private PlayerCar p2vehicle2;
        private PlayerCar p2vehicle3;
        private CarAI botCar;

        public int P1currentVehicle { get => p1currentVehicle; set =>
p1currentVehicle = value; }
        public int P2currentVehicle { get => p2currentVehicle; set =>
p2currentVehicle = value; }
        public Track Track { get => track; set => track = value; }
        public PlayerCar P1vehicle1 { get => p1vehicle1; set => p1vehicle1 =
value; }
        public PlayerCar P1vehicle2 { get => p1vehicle2; set => p1vehicle2 =
value; }
        public PlayerCar P1vehicle3 { get => p1vehicle3; set => p1vehicle3 =
value; }
        public PlayerCar P2vehicle1 { get => p2vehicle1; set => p2vehicle1 =
value; }
        public PlayerCar P2vehicle2 { get => p2vehicle2; set => p2vehicle2 =
value; }
        public PlayerCar P2vehicle3 { get => p2vehicle3; set => p2vehicle3 =
value; }
```

```
        public CarAI BotCar { get => botCar; set => botCar = value; }

        public CarSelectionScreen(IGameScreenManager gameScreenManager,Track
currentTrack,Vector2 carstartingPos)
        {
            P1vehicle1 = new PlayerCar(KeyboardKey.KEY_W, KeyboardKey.KEY_S,
KeyboardKey.KEY_A, KeyboardKey.KEY_D,KeyboardKey.KEY_LEFT_SHIFT, new
Vector2(carstartingPos.X * 180 + 180 + 90, carstartingPos.Y*180 + 90 - 10),
Program.carTexture);
            P1vehicle2 = new PlayerCar(KeyboardKey.KEY_W, KeyboardKey.KEY_S,
KeyboardKey.KEY_A, KeyboardKey.KEY_D, KeyboardKey.KEY_LEFT_SHIFT, new
Vector2(carstartingPos.X * 180 + 180 + 90, carstartingPos.Y * 180 + 90 - 10),
Program.carTexture2);
            P1vehicle3 = new PlayerCar(KeyboardKey.KEY_W, KeyboardKey.KEY_S,
KeyboardKey.KEY_A, KeyboardKey.KEY_D, KeyboardKey.KEY_LEFT_SHIFT, new
Vector2(carstartingPos.X * 180 + 180 + 90, carstartingPos.Y * 180 + 90 - 10),
Program.carTexture3);

            P2vehicle1 = new PlayerCar(KeyboardKey.KEY_UP, KeyboardKey.KEY_DOWN,
KeyboardKey.KEY_LEFT, KeyboardKey.KEY_RIGHT, KeyboardKey.KEY_RIGHT_CONTROL, new
Vector2(carstartingPos.X * 180 + 180 + 90, carstartingPos.Y * 180 + 90 + 20),
Program.carTexture);
            P2vehicle2 = new PlayerCar(KeyboardKey.KEY_UP, KeyboardKey.KEY_DOWN,
KeyboardKey.KEY_LEFT, KeyboardKey.KEY_RIGHT, KeyboardKey.KEY_RIGHT_CONTROL, new
Vector2(carstartingPos.X * 180 + 180 + 90, carstartingPos.Y * 180 + 90 + 20),
Program.carTexture2);
            P2vehicle3 = new PlayerCar(KeyboardKey.KEY_UP, KeyboardKey.KEY_DOWN,
KeyboardKey.KEY_LEFT, KeyboardKey.KEY_RIGHT, KeyboardKey.KEY_RIGHT_CONTROL, new
Vector2(carstartingPos.X * 180 + 180 + 90, carstartingPos.Y * 180 + 90 + 20),
Program.carTexture3);

            BotCar = new CarAI(new Vector2(carstartingPos.X , carstartingPos.Y ),
Program.carTexture3, currentTrack.Waypoints);
            Track = currentTrack;
            m_gameScreen = gameScreenManager;

            p1_vehicles.Add(P1vehicle1);
            p1_vehicles.Add(P1vehicle2);
            p1_vehicles.Add(P1vehicle3);

            p2_vehicles.Add(P2vehicle1);
            p2_vehicles.Add(P2vehicle2);
            p2_vehicles.Add(P2vehicle3);
        }

        public void Dispose()
        {

        }

        public void Draw()
        {
            //CAR SELECTION MENU
            Raylib.DrawTextureEx(Program.selectionP1,new Vector2( 1280 / 2 - (45
* 3 / 2) - 294,150),0f,2f,Raylib.WHITE);


            Raylib.DrawTextureEx(p1_vehicles[P1currentVehicle].CarTexture,new
Vector2(1280/2 - (45*3/2) - 300 ,720/2 - (86*3/2)),0,3,Raylib.WHITE);
            Raylib.DrawTextureEx(Program.WASD,new Vector2(1280 / 2 - (45 * 3 / 2)
- 300 -10, 720 / 2 - (86 * 3 / 2) - 200),0,1,Raylib.WHITE);
            if (P2currentVehicle != p2_vehicles.Count())
            {
                Raylib.DrawTextureEx(Program.Arrows, new Vector2(1280 / 2 - (45 *
3 / 2) + 300 - 10, 720 / 2 - (86 * 3 / 2) - 200), 0, 1, Raylib.WHITE);
            }


            //Player 1 Menu
            if (RayGui.GuiButton(new Rectangle(Program.screenwidth/2 - 125 -
300,Program.screenheight/2 , 25, 25), "<") )
            {
                if (P1currentVehicle != 0)
                {
                    P1currentVehicle --;
                }
                else
                {
                    P1currentVehicle = p1_vehicles.Count - 1;
                }

            }

            if (RayGui.GuiButton(new Rectangle(Program.screenwidth / 2 + 110 -
300,Program.screenheight / 2 , 25, 25), ">"))
            {


                if ( P1currentVehicle == p1_vehicles.Count - 1)
                {
                    P1currentVehicle = 0;
                }

                else
                {
                   P1currentVehicle ++;
                }

            }
            // player 2 menu

            if (P2currentVehicle == p2_vehicles.Count )
            {
                playCPU = true;
                Raylib.DrawTextureEx(Program.selectionCPU, new Vector2(1280 / 2 -
(45 * 3 / 2) + 305, 150), 0f, 2f, Raylib.WHITE);
                Raylib.DrawTextureEx(BotCar.CarTexture, new Vector2(1280 / 2 -
(45 * 3 / 2) + 300, 720 / 2 - (86 * 3 / 2)), 0, 3, Raylib.WHITE);

            }
            else
            {
                playCPU = false;
                Raylib.DrawTextureEx(Program.selectionP2, new Vector2(1280 / 2 -
(45 * 3 / 2) + 305, 150), 0f, 2f, Raylib.WHITE);
                Raylib.DrawTextureEx(p2_vehicles[P2currentVehicle].CarTexture,
new Vector2(1280 / 2 - (45 * 3 / 2) + 300, 720 / 2 - (86 * 3 / 2)), 0, 3,
Raylib.WHITE);
            }



            if (RayGui.GuiButton(new Rectangle(Program.screenwidth / 2 - 125 +
300, Program.screenheight / 2, 25, 25), "<"))
            {
                if (P2currentVehicle != 0)
                {
                    P2currentVehicle--;
                }
                else
                {
                    P2currentVehicle = p2_vehicles.Count;
                }

            }

            if (RayGui.GuiButton(new Rectangle(Program.screenwidth / 2 + 110 +
300, Program.screenheight / 2, 25, 25), ">"))
            {


                if (P2currentVehicle == p2_vehicles.Count)
                {
                    P2currentVehicle = 0;
                }

                else
                {
                    P2currentVehicle++;
                }

            }





            if (RayGui.GuiButton(new Rectangle(1280/2 - 100,650, 200,75),"SELECT
CARS"))
            {
                if (playCPU)
                {
                    m_gameScreen.PushScreen(new GameScreen(Track,
p1_vehicles[P1currentVehicle], p2_vehicles[P2currentVehicle - 1], m_gameScreen,
BotCar, playCPU));

                }
                else
                {
                    m_gameScreen.PushScreen(new GameScreen(Track,
p1_vehicles[P1currentVehicle], p2_vehicles[P2currentVehicle], m_gameScreen,
BotCar, playCPU));
                }

            }
        }

        public void HandleInput()
        {
            if (RayGui.GuiButton(new Rectangle(Program.screenwidth -
50,25,25,25),"x") == true)
            {
                m_gameScreen.PopScreen();
            }
        }
        public void Update()
        {

        }
    }
}
```

3.19 GameScreen.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Raylib_CsLo;
using System.Numerics;
using System.Threading;
using System.Data;
using System.Reflection.Metadata.Ecma335;
using System.Security.Cryptography.X509Certificates;
using Raylib_CsLo.InternalHelpers;
using System.Timers;
using System.Collections;
using System.ComponentModel;
using Microsoft.Toolkit.HighPerformance;

namespace NEA.Screens
{
    public class GameScreen : IGameScreen
    {

        private IGameScreenManager m_screenManager;
        private Track gameTrack;
        private PlayerCar gameCar;
        private Rectangle rect2;
        private CarAI p2Bot;
        private PlayerCar p2Car;
        private bool playCPU;
        private Vector2 p2position;
        private CheckpointManager checkpointManager;
        public List<Checkpoint> checkpointList;
        // private CarAI botCar;
        private Vector2 gridPosition;
        private Camera2D p1camera;
        private Camera2D p2camera;
        private Rectangle startcellRect;

        public RenderTexture screenCamera1 = Raylib.LoadRenderTexture(1920 / 2,
1080);
        public RenderTexture screenCamera2 = Raylib.LoadRenderTexture(1920 / 2,
1080);
        private Rectangle splitScreenRect;
        static double startingTime;
        private double p1startingTime;
        private double p2startingTime;
        private double p1currentTime;
        private double p2currentTime;
        private double p1bestLap = -1;
        private double p2bestLap = -1;

        public const int TrackXOffset = 180;
        public const float cellScale = 1.60714286f;
        public const int scaledCellWidth = 180;
        private int laps = 5;
        private int p1lapsdone = 0;
        private int p2lapsdone = 0;

        public int Laps { get => laps; }
        public int P1lapsdone { get => p1lapsdone; set => p1lapsdone = value; }
        public int P2lapsdone { get => p2lapsdone; set => p2lapsdone = value; }


        public GameScreen(Track track, PlayerCar vehicle, PlayerCar p2Vehicle,
IGameScreenManager gameScreenManager, CarAI botCar, bool playBOT)
        {

            m_screenManager = gameScreenManager;
            gameTrack = track;
            gameCar = vehicle;
            playCPU = playBOT;
            p2Bot = botCar;
            p2Car = p2Vehicle;
            startingTime = Raylib.GetTime();



            p1startingTime = Raylib.GetTime();
            p2startingTime = Raylib.GetTime();

            if (playCPU)
            {
                p2position = new Vector2(p2Bot.Position.X, p2Bot.Position.Y);
            }
            else
            {
                p2position = new Vector2(p2Car.Position1.X, p2Car.Position1.Y);
            }

            p2camera.target = p2position;

            p1camera.target = new Vector2(gameCar.Position1.X,
gameCar.Position1.Y);




            checkpointList = gameTrack.GenerateCheckpoints();
            checkpointManager = new CheckpointManager(checkpointList, Laps);

            startcellRect = checkpointList.ElementAt(1).CheckRect;

            p1camera.zoom = 3.5f;
            p2camera.zoom = 3.5f;





            splitScreenRect = new Rectangle(0.0f, 0.0f,
(float)screenCamera1.texture.width, (float)-screenCamera1.texture.height);
```

```
            Raylib.SetWindowSize(1920, 1080);
            Raylib.ToggleFullscreen();

        }

        public void Dispose()
        {

        }

        public void Draw()
        {

            //draws split screen cars and HUD

            Raylib.BeginTextureMode(screenCamera1);
            Raylib.ClearBackground(Raylib.GREEN);
            Raylib.BeginMode2D(p1camera);
            gameTrack.TrackGrid.Draw(TrackXOffset, (int)((float)Cell.width *
cellScale), cellScale, 0);
            gameCar.Draw();

            if (playCPU)
            {
                p2Bot.Draw();
            }
            else
            {
                p2Car.Draw();
            }
            Raylib.EndMode2D();

            Raylib.EndTextureMode();


            Raylib.BeginTextureMode(screenCamera2);
            Raylib.ClearBackground(Raylib.GREEN);
            Raylib.BeginMode2D(p2camera);

            gameTrack.TrackGrid.Draw(TrackXOffset, (int)((float)Cell.width *
cellScale), cellScale, 0);
            gameCar.Draw();
            // Initialize AI position or second player position based on user
choice in the previous screen
            if (playCPU)
            {
                p2Bot.Draw();
            }
            else
            {
                p2Car.Draw();
            }

            Raylib.EndMode2D();

            Raylib.EndTextureMode();


            Raylib.BeginDrawing();
            Raylib.ClearBackground(Raylib.BLACK);
            Raylib.DrawTextureRec(screenCamera1.texture, splitScreenRect, new
Vector2(0, 0), Raylib.WHITE);
```

```
            Raylib.DrawTextureRec(screenCamera2.texture, splitScreenRect, new
Vector2(1980 / 2 - 25, 0), Raylib.WHITE);

            Raylib.DrawText("BEST LAP:" + MathF.Round((float)p2bestLap, 1), 1980
- 285, 125, 30, Raylib.BLACK);
            Raylib.DrawText("BEST LAP:" + MathF.Round((float)p1bestLap, 1), 1980
/ 2 - 260, 125, 30, Raylib.BLACK);

            Raylib.DrawTexture(Program.Laps, 1980 / 2 - 250, 75, Raylib.WHITE);
            Raylib.DrawText(P1lapsdone + "/" + Laps, 1980 / 2 - 250 + 85, 82, 30,
Raylib.BLACK);
            Raylib.DrawTexture(Program.Laps, 1980 - 275, 75, Raylib.WHITE);
            Raylib.DrawText(P2lapsdone + "/" + Laps, 1980 - 250 + 85 - 25, 82,
30, Raylib.BLACK);

            Raylib.DrawTexture(Program.lapTime, 1980 / 2 - 250, 25,
Raylib.WHITE);
            Raylib.DrawText(Math.Round(p1currentTime, 2).ToString(), 1980 / 2 -
250 + 97, 32, 30, Raylib.BLACK);
            Raylib.DrawTexture(Program.lapTime, 1980 - 275, 25, Raylib.WHITE);
            Raylib.DrawText(Math.Round(p2currentTime, 2).ToString(), 1980 - 275 +
97, 32, 30, Raylib.BLACK);

            Raylib.DrawRectangle(1980 / 4 - 120, 950, (int)gameCar.Boost * 3, 75,
Raylib.YELLOW);
            Raylib.DrawTextureEx(Program.boostFrame, new Vector2(1980 / 4 - 200,
700), 0, 4, Raylib.WHITE);
            if (playCPU != true)
            {
                Raylib.DrawRectangle(1980 / 2 + 1980 / 4 - 120, 950,
(int)p2Car.Boost * 3, 75, Raylib.YELLOW);
                Raylib.DrawTextureEx(Program.boostFrame, new Vector2(1980 / 2 +
1980 / 4 - 200, 700), 0, 4, Raylib.WHITE);
            }









        }

        //Checks when a lap is completed by each of the two players
        private void LapCollisions()
        {

            //Checks if the first player is playing a bot or another user to
decide which rectangle to use for collisions
            if (playCPU)
            {
                rect2 = p2Bot.CarRect;
            }
            else
            {
                rect2 = p2Car.PlayerRect;
            }
            //checks for checkpoint collisions
            if (checkpointManager.P1Queue.Count() != 0 &&
Raylib.CheckCollisionRecs(gameCar.PlayerRect,
checkpointManager.P1Queue.First().CheckRect))
```

```
            {

                checkpointManager.p1CheckpointPassed();


            }
            if (checkpointManager.P2Queue.Count() != 0 &&
Raylib.CheckCollisionRecs(rect2, checkpointManager.P2Queue.First().CheckRect))
            {


                checkpointManager.p2CheckpointPassed();


            }

            //checks if the lap is completed and if all the laps are completed by
one the user shows the winner screen
            if (P2lapsdone == Laps && Raylib.CheckCollisionRecs(rect2,
startcellRect) && checkpointManager.p2IsLapValid() == true)
            {
                //Raylib.ToggleFullscreen();
                m_screenManager.PushScreen(new WinnerScreen("P2",
m_screenManager));

                Raylib.SetWindowSize(1280, 720);
            }
            else if (P1lapsdone == Laps &&
Raylib.CheckCollisionRecs(gameCar.PlayerRect, startcellRect) &&
checkpointManager.p1IsLapValid() == true)
            {
                m_screenManager.PushScreen(new WinnerScreen("P1",
m_screenManager));

                //Raylib.ToggleFullscreen();
                Raylib.SetWindowSize(1280, 720);
            }
            else if (P2lapsdone != Laps && Raylib.CheckCollisionRecs(rect2,
startcellRect) && checkpointManager.p2IsLapValid() == true)
            {
                P2lapsdone++;
                if (p2currentTime <= p2bestLap || p2bestLap < 0)
                {
                    p2bestLap = p2currentTime;
                }
                p2startingTime = p1currentTime + p1startingTime;

            }

            else if (P1lapsdone != Laps &&
Raylib.CheckCollisionRecs(gameCar.PlayerRect, startcellRect) &&
checkpointManager.p1IsLapValid() == true)
            {
                P1lapsdone++;
                if (p1currentTime <= p1bestLap || p1bestLap < 0)
                {
                    p1bestLap = p1currentTime;
                }
                p1startingTime = p1currentTime + p1startingTime;
            }

        }
        /// <summary>
        /// Collision between the two cars
        /// </summary>
        private void Collision()
        {
            //Checks if the first player is playing a bot or another user to
decide which rectangle to use for collisions
            if (playCPU)
            {
                rect2 = p2Bot.CarRect;
            }
            else
            {
                rect2 = p2Car.PlayerRect;
            }

            if (Raylib.CheckCollisionRecs(gameCar.PlayerRect, rect2))
            {

                Vector2 p1Origin = new Vector2(gameCar.PlayerRect.x +
gameCar.PlayerRect.width / 2, gameCar.PlayerRect.y + gameCar.PlayerRect.height /
2);
                Vector2 p2Origin = new Vector2(rect2.x + rect2.width / 2, rect2.y
+ rect2.height / 2);
                Vector2 originVec = Vector2.Normalize(p1Origin - p2Origin);
                gameCar.CarVelocity += originVec;
                p2Car.CarVelocity -= originVec;

            }


        }
        /// <summary>
        /// Collision between cars and the Map boundaries
        /// </summary>
        public void MapLimits()
        {

            Vector2 p1Origin = new Vector2(gameCar.PlayerRect.x +
gameCar.PlayerRect.width / 2, gameCar.PlayerRect.y + gameCar.PlayerRect.height /
2);
            Vector2 p2Origin = new Vector2(p2Car.PlayerRect.x +
p2Car.PlayerRect.width / 2, p2Car.PlayerRect.y + p2Car.PlayerRect.height / 2);

            Vector2.Normalize(p1Origin);



            if (Raylib.CheckCollisionRecs(gameCar.PlayerRect, new Rectangle(0, 0,
190, 1080)) == true)
            {

                gameCar.CarVelocity = new Vector2(gameCar.CarVelocity.X + 3,
gameCar.CarVelocity.Y);
            }
            if (Raylib.CheckCollisionRecs(gameCar.PlayerRect, new Rectangle(1790,
0, 120, 1080)) == true)
            {
                gameCar.CarVelocity = new Vector2(gameCar.CarVelocity.X - 3,
gameCar.CarVelocity.Y);

            }
            if (Raylib.CheckCollisionRecs(p2Car.PlayerRect, new Rectangle(0, 0,
190, 1080)) == true)
            {

                p2Car.CarVelocity = new Vector2(p2Car.CarVelocity.X + 3,
p2Car.CarVelocity.Y);

            }
            if (Raylib.CheckCollisionRecs(p2Car.PlayerRect, new Rectangle(1790,
0, 120, 1080)) == true)
            {
                p2Car.CarVelocity = new Vector2(p2Car.CarVelocity.X - 3,
p2Car.CarVelocity.Y);
            }


            if (Raylib.CheckCollisionRecs(gameCar.PlayerRect, new Rectangle(0, 0,
1920, 10)) == true)
            {
                gameCar.CarVelocity = new Vector2(gameCar.CarVelocity.X,
gameCar.CarVelocity.Y + 3);
            }
            if (Raylib.CheckCollisionRecs(gameCar.PlayerRect, new Rectangle(0,
1080 - 10, 1920, 190)) == true)
            {
                gameCar.CarVelocity = new Vector2(gameCar.CarVelocity.X,
gameCar.CarVelocity.Y - 3);
            }


            if (Raylib.CheckCollisionRecs(p2Car.PlayerRect, new Rectangle(0, 0,
1920, 10)) == true)
            {
                p2Car.CarVelocity = new Vector2(p2Car.CarVelocity.X,
p2Car.CarVelocity.Y + 3);
            }
            if (Raylib.CheckCollisionRecs(p2Car.PlayerRect, new Rectangle(0, 1920
- 190, 1920, 190)) == true)
            {
                p2Car.CarVelocity = new Vector2(p2Car.CarVelocity.X,
p2Car.CarVelocity.Y - 3);
            }


        }
        public void HandleInput()
        {

        }




        public void Update()
        {




            if (playCPU)
            {
                p2position = new Vector2(p2Bot.Position.X, p2Bot.Position.Y);
            }
            else
            {
                p2position = new Vector2(p2Car.Position1.X, p2Car.Position1.Y);
            }

            p1camera.target = new Vector2(gameCar.Position1.X,
gameCar.Position1.Y);
            p1camera.offset = new Vector2(1920 / 4, 1080 / 2);
            p2camera.target = p2position;
            p2camera.offset = new Vector2(1920 / 4, 1080 / 2);

            gameCar.Update();
            if (playCPU)
            {
                p2Bot.Update();
            }
            else
            {
                p2Car.Update();
            }

            Collision();
            LapCollisions();
            p1currentTime = Raylib.GetTime() - p1startingTime;
            p2currentTime = Raylib.GetTime() - p2startingTime;
            MapLimits();
            p1GroundCollision();
            p2GroundCollision();

        }



        /// <summary>
        /// Calculates the corrispondent cell in the grid from the position in
the screen, and returns it
        /// </summary>
        /// <param name="position"></param>
        /// <returns></returns>
        public Vector2 PosToGrid(Vector2 position)
        {

            position.X = (int)((position.X - 180) / 180);
            position.Y = (int)((position.Y) / 180);
            return position;
        }



        /// <summary>
        /// Checks if player 1 goes out of the road and apply braking in case
        /// </summary>
        public void p1GroundCollision()
        {
            Vector2 temp;
            temp = PosToGrid(gameCar.Position1);
            if (gameTrack.TrackGrid.GridArray[(int)temp.X, (int)temp.Y].Road ==
0)
            {
                gameCar.CarVelocity = new Vector2(gameCar.CarVelocity.X * (1.0f -
gameCar.CarBrake), gameCar.CarVelocity.Y * (1.0f - gameCar.CarBrake));
            }
        }

        /// <summary>
        /// Checks if player 2 goes out of the track and in case apply braking
        /// </summary>
        public void p2GroundCollision()
        {
            Vector2 temp;
            temp = PosToGrid(p2Car.Position1);
            if (gameTrack.TrackGrid.GridArray[(int)temp.X, (int)temp.Y].Road ==
0)
            {


                p2Car.CarVelocity = new Vector2(p2Car.CarVelocity.X * (1.0f -
p2Car.CarBrake), p2Car.CarVelocity.Y * (1.0f - p2Car.CarBrake));
            }

        }




    }

}
```

3.20 WinnerScreen.cs

```
using Raylib_CsLo;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NEA.Screens
{
    public class WinnerScreen : IGameScreen
    {
        private string winner;
        private IGameScreenManager m_screenManager;
        public WinnerScreen(string Winner, IGameScreenManager m_screenManager1)
        {
            winner = Winner;
            m_screenManager = m_screenManager1;
            Raylib.ToggleFullscreen();
        }

        public void Dispose()
        {

        }

        public void Draw()
        {
            Raylib.DrawRectangle(Program.screenwidth / 2 - 210,
Program.screenheight / 2 - 100, 405, 100,Raylib.LIGHTGRAY);
            Raylib.DrawText(winner + " WON", Program.screenwidth / 2 - 200,
Program.screenheight / 2 - 100, 100, Raylib.BLACK);
            if (RayGui.GuiTextBox(new Rectangle(Program.screenwidth / 2 - 210,
Program.screenheight / 2 - 100, 410, 100),"", 700, false) != false)
```

```
            {
                m_screenManager.PopScreen();
                m_screenManager.PopScreen();
                m_screenManager.PopScreen();

            }

        }

        public void HandleInput()
        {

        }

        public void Update()
        {

        }
    }
}
```

3.21 GameScreenManager.cs

```
using Raylib_CsLo;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using static NEA.Screens.GameScreenManager;

namespace NEA.Screens
{

    /// <summary>
    /// This class manages the list of game screens for the application.
    /// </summary>
    public class GameScreenManager : IGameScreenManager
    {

        private readonly List<IGameScreen> m_gameScreens = new
List<IGameScreen>();

        public GameScreenManager()
        {

        }



        //Pushes a new game screen onto the top of the list
        public void PushScreen(IGameScreen screen)
        {

            m_gameScreens.Add(screen);

        }

        //Checks if the game screen list is empty
        private bool IsScreenListEmpty
        {
```

```
    get { return m_gameScreens.Count <= 0; }
}
//returns the current game screen
private IGameScreen GetCurrentScreen()
{
    return m_gameScreens[m_gameScreens.Count - 1];
}

//Removes all screens
private void RemoveAllScreens()
{
    while (IsScreenListEmpty != true)
    {
        RemoveCurrentScreen();
    }
}
//removes current screen
private void RemoveCurrentScreen()
{
    var screen = GetCurrentScreen();
    screen.Dispose();
    m_gameScreens.Remove(screen);
}
// pops the current screen off the list
public void PopScreen()
{
    if (!IsScreenListEmpty)
    {
        RemoveCurrentScreen();

    }
    if (!IsScreenListEmpty)
    {
        var screen = GetCurrentScreen();

    }
}

public void HandleInput()
{
    if (!IsScreenListEmpty)
    {
        var screen = GetCurrentScreen();

        screen.HandleInput();

    }
}
//Update current scrknown
public void Update()
{
    if (!IsScreenListEmpty)
    {
        var screen = GetCurrentScreen();

        screen.Update();

    }
}
//draws current screen
public void Draw()
{
    if (!IsScreenListEmpty)
```

```
            {
                var screen = GetCurrentScreen();

                screen.Draw();

            }
        }

        public void Dispose()
        {
            RemoveAllScreens();
        }


    }



}
```

3.22 ScreenInterfaces.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NEA.Screens
{
    public interface IGameScreen
    {

        void Update();
        void Draw();
        void Dispose();
        void HandleInput();

    }

    public interface IGameScreenManager
    {

        void PushScreen(IGameScreen screen);
        void PopScreen();


        void Update();
        void Draw();
        void HandleInput();


    }
}
```

3.23 RaylibBindingsConverter.cs

```
using Microsoft.Toolkit.HighPerformance.Buffers;
using Raylib_CsLo;
using Raylib_CsLo.InternalHelpers;
using System.Numerics;
using System.Runtime.InteropServices;

internal static class RaylibBindingsConverter
{

    internal static Raylib_CsLo.Rectangle ConvertRectangle(Rectangle rectangle)
    {
        return new Raylib_CsLo.Rectangle(rectangle.x, rectangle.y,
rectangle.width, rectangle.height);
    }



    internal unsafe static string ConvertSbyte(sbyte* sbytes, int maxLength)
    {
        if (sbytes is null)
        {
            return string.Empty;
        }

        int length = 0;

        // Find the length of the string, up to maxLength
        while (length < maxLength && sbytes[length] != 0)
        {
            length++;
        }

        // Create a byte array and copy the sbytes into it
        byte[] bytes = new byte[length];
        Marshal.Copy((IntPtr)sbytes, bytes, 0, length);

        // Convert the byte array to a string
        string str = System.Text.Encoding.UTF8.GetString(bytes);

        return str;
    }



}
```

3.24 GUI.CS

```
using Microsoft.Toolkit.HighPerformance.Buffers;
using Raylib_CsLo;
using Raylib_CsLo.InternalHelpers;
using System.Numerics;
using System.Text;
using static RaylibBindingsConverter;

namespace NEA
{
    public static class Gui
    {



        public static int GuiTextInputBox(Rectangle bounds, string? title,
string? message, string? buttons, ref string? text, int textMaxSize = 255)
        {
            title ??= string.Empty;
            message ??= string.Empty;
            buttons ??= string.Empty;
            text ??= string.Empty;
            unsafe
            {
                SpanOwner<sbyte> spanOwner = title.MarshalUtf8();
                try
                {
                    SpanOwner<sbyte> spanOwner2 = message.MarshalUtf8();
                    try
                    {
                        SpanOwner<sbyte> spanOwner3 = buttons.MarshalUtf8();
                        try
                        {
                            SpanOwner<sbyte> spanOwner4 = text.MarshalUtf8();
                            try
                            {
                                var textBytes = spanOwner4.AsPtr();
                                int buttonClicked =
Raylib_CsLo.RayGui.GuiTextInputBox(ConvertRectangle(bounds), spanOwner.AsPtr(),
spanOwner2.AsPtr(), spanOwner3.AsPtr(), textBytes, textMaxSize, null);
                                text = ConvertSbyte(textBytes, textMaxSize);
                                return buttonClicked;
                            }
                            finally
                            {
                                spanOwner4.Dispose();
                            }
                        }
                        finally
                        {
                            spanOwner3.Dispose();
                        }
                    }
                    finally
                    {
                        spanOwner2.Dispose();
                    }
                }
                finally
                {
                    spanOwner.Dispose();
                }
            }
        }

    }
}
```

4 TESTING

Requirement ID	**Test description**	**Input test data**	**Expected Outcome**	**Timestamp**	**Result**
1.1	Run the game	N/A	The starfield animation will play on screen, the stars are spawned randomly in the screen and they move towards the player		
1.1.2	Moving the mouse on the screen and placing it on top of the play button	Mouse movement	The stars keep move progressively faster accordingly to the mouse x position, since the play button is placed on the top right position screen, the more the mouse gets closer to the button the speeds increases		
1.2 and 1.2.1	Press on the “PLAY” button	Left click	When the “PLAY ”button is pressed, the user gets brought to the Track Selection screen		
1.2.2	Press the ESC key	ESC key	The game get closed and stop running when the ESC key is pressed		
2 and 2.1	Open track selection screen	N/A	User will be shown the Track Selection screen containing the “SELECT TRACK”, “TRACK EDITOR”, “DELETE TRACK” buttons and two arrows buttons		

Requirement ID	**Test description**	**Input test data**	**Expected Outcome**	**Timestamp**	**Result**
2.2	Open track selection screen	N/A	The screen shows a preview of the look of the current track by reading the content of the binary file where we the tracks are stored		
2.2.1	Press the “SELECT TRACK” button	Left click	When “SELECT TRACK” is pressed the user gets brought to the Car Selection Screen		
2.2.2	Press the “TRACK EDITOR” button	Left click	When “TRACK EDITOR” is pressed the user gets brough to the Track Editor Screen		
2.2.3	Press the “DELETE TRACK BUTTON”	Left click	When “DELETE TRACK” is pressed the current track showed in the menu gets deleted from the list of tracks of the game		
2.2.4 and 2.2.5	Press one of the arrows buttons	Left click	When the left or right arrow are pressed, we scroll through the list of different tracks available and so it changes the current track. every time we press one of the arrow it will update if more that one track is in the list.		
2.2.5.1	Press “DELETE TRACK” to remove all the tracks from the list	N/A	If the list of tracks in the game is empy the track preview won’t be shown as there is no current track		

Requirement ID	**Test description**	**Input test data**	**Expected Outcome**	**Timestamp**	**Result**
2.2.6	Press one of the arrows buttons	Left click	The Menu shows on top of the preview of the current track his name by reading the content of the binary file where the tracks are stored, so every time we press one of the arrows it will update if more than one track is present in the game.		
2.2.7	Press the ESC key	ESC key	The user will exit the game if the ESC key on the keyboard is pressed.		
3. and 3.1 and 3.2	Open Car Selection screen	N/A	User will be shown the Car Selection screen and this screen will feature 4 arrows buttons, 2 buttons ("X", "SELECT CARS") and it will show the image of the current selected car for the first and second player.		
3.3.1	Press one of the two arrows buttons in the left side of the screen	Left click	When one of the two arrows in the left side of the screen are pressed, we change the current car of the first player .		
3.3.2	Press one of the two arrows buttons in the right side of the screen.	Left click	When one of the two arrows in the right side of the screen are pressed, we change the current car of the second player.		

Requirement ID	**Test description**	**Input test data**	**Expected Outcome**	**Timestamp**	**Result**
3.3.3	Press one of the arrows buttons.	Left click	When one of the arrows is pressed the image showing the previous car texture will update to the current one.		
3.3.4	Press the arrows button to check if when the cpu car is selected the commands are not shown.	Left click	Relative Commands for player 1 and 2 should be shown respectively above the two cars, if the user choose to play against the cpu no commands will be shown above the right side car		
3.3.5	Press the “SELECT CAR” button.	Left click	When “SELECT CAR” is pressed, the user will be taken to the Gameplay screen.		
3.3.6	Press the “X” button.	Left click	When “X” is pressed, the user will go back to the Track Selection screen		
4 and 4.1	Open Track Editor screen.	N/A	User will be shown the Track Editor featuring an interactive 9x6 cells grid, 3 buttons (“SAVE”, “ERASE” and “X”) and a text box.		
4.1.1 and 4.1.1.1	Press the left mouse button on an empty or a straight road grid cell.	Left click	When the user left clicks on an empty cell or a straight road cell, a curve texture will be inserted into the cell.		
4.1.1 and 4.1.1.2	Press the right mouse button on an empty or curve road grid cell.	Right click	When the user right clicks on an empty cell or a curve texture cell, a straight road texture will be inserted into the cell.		

Requirement ID	**Test description**	**Input test data**	**Expected Outcome**	**Timestamp**	**Result**
4.1.1 and 4.1.1.3	Press the left mouse button on a curve road grid cell.	Left click	When the user left clicks on a curve cell, it will rotate the texture by 90°.		
4.1.1 and 4.1.1.4	Press the right mouse button on a straight road grid cell.	Right click	When the user right clicks on a straight road cell, it will rotate the texture by 90°.		
4.1.1 and 4.1.1.5	Press the middle mouse button on a grid cell.	Middle mouse button	When the user middle clicks on any sort of road cell it gets set as a empty grass cell.		
4.1.1 and 4.1.1.6	Press “ENTER” key.	ENTER key	When the user press Enter on the Keyboard, the empty cells of the grid will be filled by a grass texture.		
4.1.2	Press the “SAVE” key after creating a valid track.	Left click	When “SAVE” is pressed and the track is valid, the track created in the grid will be saved with the name inserted in the text box in a binary file and the user will be brought to the Track Selection screen		
4.1.3	Press the “SAVE” key after creating a non-valid track, and after press on the error message window.	Left click	When ”SAVE” is pressed and the track is not valid, a error message will be shown, and if pressed it will disappear leaving the user to the track editor and set every cell of the grid to be an empty cell.		

Requirement ID	**Test description**	**Input test data**	**Expected Outcome**	**Timestamp**	**Result**
4.1.4	Press the “ERASE” button.	Left click	When “ERASE” is pressed, every cell of the grid gets substituted with an empty cell.		
4.1.5	Press the “X” button.	Left click	When “X” is pressed, the user will go back to the Track Selection screen.		
5 and 5.1 and 5.2	Open Game screen.	N/A	User will be shown the Game screen. In this screen the users will be able to play the game,it shows 2 different screens and the two players cameras.		
5.1.1	Press “W” key.	W key	If “W“ is pressed the first player car accelerates forward.		
5.1.2	Press “S” key.	S key	if “S” is pressed the first player car brakes.		
5.1.3	Press “A” key	A key	if “A” is pressed the first player car rotates anti clock wise and the first player boost recharges.		
5.1.4	Press “D” key	D key	if “D” is pressed the second player car rotates clock wise and the first player boost recharges.		
5.1.5	Press the SHIFT key	SHIFT key	if the SHIFT key is used when the boost bar is not empty the first car player uses its boost		

Requirement ID	Test description	Input test data	Expected Outcome	Timestamp	Result
5.1.6	Press the UP ARROW key	UP ARROW key	if the UP ARROW key is pressed the second player car accelerates forward.		
5.1.7	Press the DOWN ARROW key	DOWN ARROW key	if The DOWN ARROW key is pressed the second player car brakes.		
5.1.8	Press the LEFT ARROW key	LEFT ARROW key	if the LEFT ARROW key is pressed the second player car rotates anti clock wise and the second player boost recharges.		
5.1.9	Press the RIGHT ARROW key	RIGHT ARROW key	if the RIGHT ARROW key is pressed the second player car rotates clock wise and the second player boost recharges.		
5.1.10	Press the CONTROL key	CONTROL key	if the right CONTROL key is used and the second player boost bar is not empty the second car player uses its boost.		
5.3.1 and 5.3.2 and 5.3.3 and 5.3.4 and 5.3.5	Open Game screen and move cars.	W/A/S/D keys or ARROWS keys	The left half of the screen will show on his right top side the laps percurred by the first player, the lap time of the first player, the best lap time registered by the first player in this race, the boost		

Requirement ID	Test description	Input test data	Expected Outcome	Timestamp	Result
			bar of the first player and the camera will follow the second player car.		
5.4.1 and 5.4.2 and 5.4.3 and 5.4.4 and 5.4.5	Open Game screen and move cars.	W/A/S/D keys or ARROWS keys	The right half of the screen will show on his right top side the laps percurred by the second player, the lap time of the second player, the best lap time registered by the second player in this race, the boost bar of the second player and the camera will follow the second player car.		
5.5	One of the two players completes a valid lap of the track	W/A/S/D keys or ARROWS keys	When one player completes a valid lap the lap time gets reset, the best lap time gets updated if the lap time is better than the previously registered one, and the completed laps number gets updated.		
5.6	One of the two players finishes the race	W/A/S/D keys or ARROWS keys	When one Player completes all the laps in the race User will be shown the Winner Screen.		
6 and 6.1	Open Winner Screen.		User will be shown the Winner Screen and in it P1 WON or P2 WON will be shown accordingly on who won the race.		

Requirement ID	**Test description**	**Input test data**	**Expected Outcome**	**Timestamp**	**Result**
6.2	Press the button with the name of the winner.	Left click	When the button with the name of the winner is pressed the game will bring the user back to the Track Selection screen.		