Project 3 / 2D Game With Unity

“Highway Havoc”

COMP 204 Programming Studio

Report prepared by:

Meryem Rana TÜRKER 042001021

Berat KAYA 042001046

Yağmur PARMAKSIZ 04200154

Date: 5 June 2023

Abstract

Highway Havoc is a 2D traffic game designed with Unity programming. The game replicates a real-world traffic situation where the player must drive a car on a congested roadway and avoid hitting other drivers. The keyboard's arrow keys are used to control the game. The forward key accelerates while the left and right keys are used to alter direction.

Additionally, the player has the option of selecting a vehicle. The main focus of the game is to gain points while driving an automobile through traffic. The game prioritizes visual appeal, and it has a point accumulation system. Additionally, there are options to increase speed and score more points.

Keywords: Highway Havoc, 2D traffic game, Unity programming, realistic traffic scenario, control vehicle, change direction, choose vehicle etc.

Overview About Unity and 2D Games

Unity is one of the most used game engines by game developers. Unity is configured for a variety of game genres and is therefore frequently used by game developers. Unity offers many tools and features in the game development process. These features include 2D and 3D graphics, a physics engine, animation tools, a scene editor for managing game objects, and a code editor for game mechanics. With these features, developers can easily create and edit their games.

Visual Studio Code (VSCode) is a code editor often used by game developers. VSCode can be used with Unity and allows game developers to edit and debug Unity projects. You can set Unity, VSCode as default code editor. Setting up the connection between Unity and VSCode is pretty easy. First, you need to download and install VSCode. Next, go to the "Edit" menu in Unity and click "Preferences". Go to the "External Tools" section and set the "External Script Editor" option to VSCode. After making this adjustment, VSCode will automatically open any code file in Unity.

In game development, the combined use of Unity and VSCode allows developers to develop and debug their games. In addition to the features that Unity provides, VSCode gives developers more options when writing code and makes the coding process easier.

As a result, the combination of Unity and VSCode is very useful in the game development process. These tools allow developers to create and edit their games faster and more efficiently.

Figure 1: Some examples of 2D Games

1.About the Project

1.1. Description of the Project

In this project, we have developed a game that includes some popular mechanics of basic 2D car games and additional features such as car acceleration. Our team developed an engaging 2D car game that incorporates popular mechanics from classic car games while introducing exciting new features, such as car acceleration. The primary objective of the game is to navigate through a dynamic road while striving to achieve a high score without colliding with other cars. Drawing inspiration from the well-known "Traffic Racer" game, we have created an immersive gaming experience by implementing various gameplay elements.

One of the key highlights of our game is the non-player character (NPC) cars that players encounter on their journey. Overtaking these cars requires precise timing and skillful maneuvering. Players must exercise caution and remain vigilant while navigating through the bustling traffic. To further enhance the gameplay experience, we have introduced the ability for players to accelerate their cars, allowing them to traverse the road at higher speeds. This feature adds an exhilarating element to the game, requiring players to balance speed with caution, as increased velocity can also amplify the risk of collisions.

Our team dedicated considerable effort to ensure the visual appeal of the game. While we have utilized certain graphic model assets from video sources on the internet and udemy, the majority of the game's art assets were meticulously created. This approach allowed us to maintain a consistent and cohesive visual style throughout the game, enhancing its immersive qualities. The

primary target of this project was to deliver a high quality Unity game that captivates players with its immersive concept and engaging gameplay mechanics. We have strived to create an enjoyable gaming experience that combines familiar elements from classic car games with innovative features, resulting in a gameplay experience for players of all ages.

Figure 2: Logo of the Game

2. Solution

We designed a car game that combines the classic features of a 2D game with innovative additions developed by us. We have meticulously structured the game's source code to ensure a well-organized and efficient development process. Classes and methods in the codebase are separated according to their specific functions and tasks for easier maintenance and scalability. To further improve the quality of the game, we've created an extensive class hierarchy within Unity that helps us implement a well thought-out architecture. (Figure 2)

This situation provides a structured implementation of the game's functions. This results in smooth gameplay and makes it easy to add new features in the future. By adopting this systematic approach, we aimed to provide an immersive gaming experience to our players.

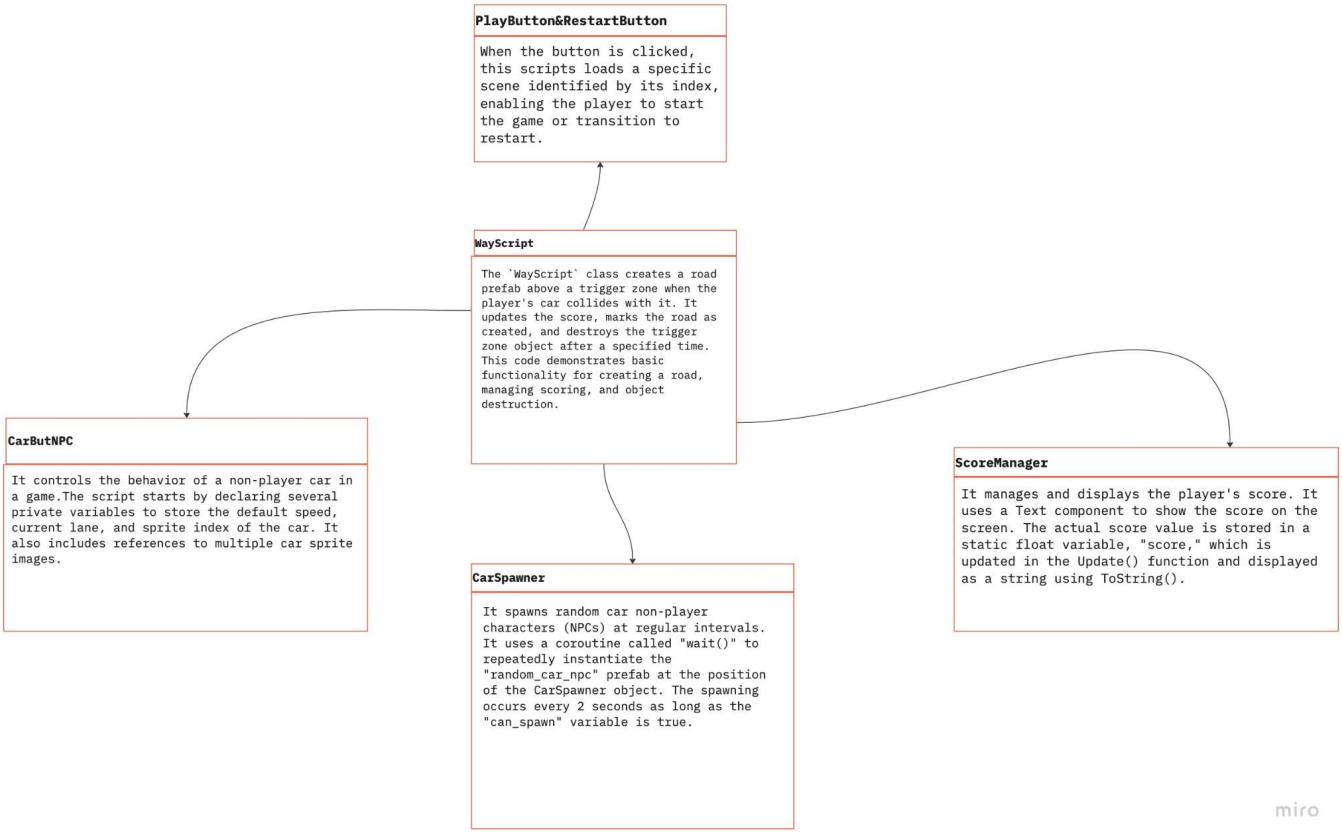


Figure 3: Class Hierarchy Diagram of the Program

Figure 4: Graphical User Interface of the Program

2.1. Implementation of The Program

"Highway Havoc" car game covers all the basic features one would expect from a simple game mechanic. To provide a streamlined development process, we have structured the program using 7 C# classes that adhere to object-oriented programming principles and use effective game design techniques. These classes work in harmony to bring the game to life, providing a cohesive and immersive experience. Every event in the game is somehow dependent on the current state of the game. So an important aspect of the functionality of the game is in its dynamic part. This interdependence creates an uninterrupted flow within the game cycle, where actions and outcomes unfold based on the player's interactions and progress. The reason we implemented this flow was to provide players with an immersive experience that allows them to focus on the mechanics and challenges of the game. The design of these classes within the coding keeps the game running smoothly while also allowing for future expansions and addition of new features to keep the game fresh.

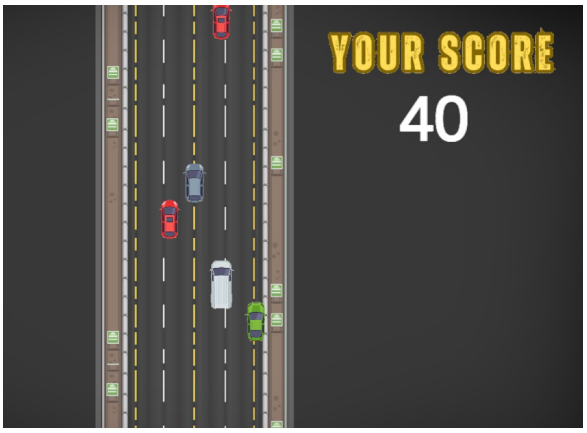


Figure 4: Game Screen of The Game

2.1.1. WayScript Class

This class includes the road dynamics in the car game. The script is attached to a game object and is responsible for creating a new game object when another game object with the tag "mainCar" collides with it. The script keeps track of whether the path is made (stored in the "way_done" variable) and ensures that the path is created only once.

When the collision occurs and the path is not yet built, the script calculates the occurrence location for the new path object. Adds 10 units to the script's current position on the y-axis of the game object and sets the z-axis to 0 to keep the same plane. The new path object is then initialized at the calculated spawn position, with no return. After creating the path, the script sets the "way_done" variable to true to indicate that the path has been made. It also adds 10 to the "score" variable in the "ScoreManager" class, which tracks the game's score. Finally, the script schedules the current game object to be destroyed after 6 seconds. This object is the object that the script is attached to in the Unity system.

Generally, this script allows a road object to be created when the "mainCar" collides with it, keeping track of the road's construction status and performing other related actions such as updating the score.

Figure 5: The Methods of WayScript Class

2.1.2. CarButNPC Class

This class is a script written to control non-player character NPC cars in our game. The script is added to every NPC car object in the game.

In the "Start()" method, various initializations and randomizations occur. The script takes the necessary components like "SpriteRenderer" and "Rigidbody2D" and assigns them to the corresponding variables. It also generates a random value between 1 and 4 for the NPC car lane and a random value between 1 and 18 for the car sprite ("car_sprite").

Depending on the randomly selected lane, the starting position of the NPC car is determined. The position is adjusted along the y-axis to make the vehicle appear at the top of the screen ready to move down. The available strip options are predefined, and each stripe is associated with a specific x-coordinate value.

The script then assigns the appropriate sprite to the NPC car based on the randomly selected "car_sprite". This is achieved using a switch statement that maps the value "car_sprite" to the corresponding sprite variable. In the "FixedUpdate()" method, the speed of the NPC car is updated to keep it moving downward at a constant speed. Speed is determined by the variable `default_rate` and `Time.deltaTime` multiplied by 50 to ensure consistent motion at different frame rates.

The script also includes collision detection and trigger logic. When the NPC car collides with the main car (identified by the "mainCar" tag), the scene with index 2 is loaded using

`SceneManager.LoadScene(2)`. This represents game over and reboot scenario. Additionally, the "OnTriggerEnter2D()" method is called when the NPC car triggers a collider with the "Overtaking" tag. In this case, the NPC car has successfully completed the overtaking maneuver and the player's score (represented by the variable 'ScoreManager.score') is increased by 30.

Overall, this script controls the behavior of NPC cars in-game, including their movement, character assignment, collision detection, and score tracking for successful overtaking.

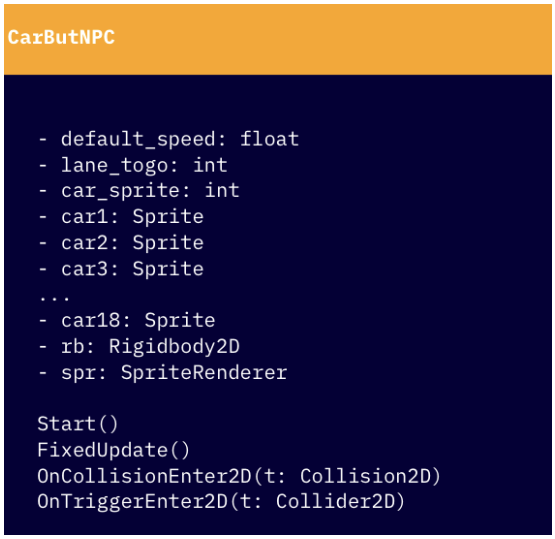


Figure 6: The Methods of CarButNPC Class

2.1.3. CarSpawner Class

This class creates a car creation object that will recursively initialize car prefabs at a given interval. The "CarSpawner" class inherits from the "MonoBehaviour" class, which is a base class for Unity scripts. There are several member variables within this class. One is a global variable named "random_car_npc" of type "GameObject" and represents the car prefab to be created.

Another is a boolean variable called "can_spawn" that determines whether car spawning is allowed. The default value is set to "true", indicating that cars can be created at startup.

The "Start()" method is a Unity callback executed when the script is started. It starts the production process by starting a coroutine named `wait()`. Coroutines are functions that can be executed on multiple frames, allowing time delays and asynchronous behavior. The "wait()" coroutine contains a while loop that continuously creates the cars as long as the "can_spawn" variable is true. Inside the loop, it uses the "Instantiate()" function to instantiate a non-return "random_car_npc" prefab at the location of the "CarSpawner" object. After sampling, it pauses the execution of the coroutine for 2 seconds using "yield return new WaitForSeconds(2f)" and creates a delay between each car spawning.

The while loop continues until can_spawn is false. When the condition is no longer met, the loop terminates and completes coroutine execution. You can change the value of the "can_spawn" variable to control the production behavior. Setting it to "false" will stop the production process, while setting it back to "true" will allow it to continue.

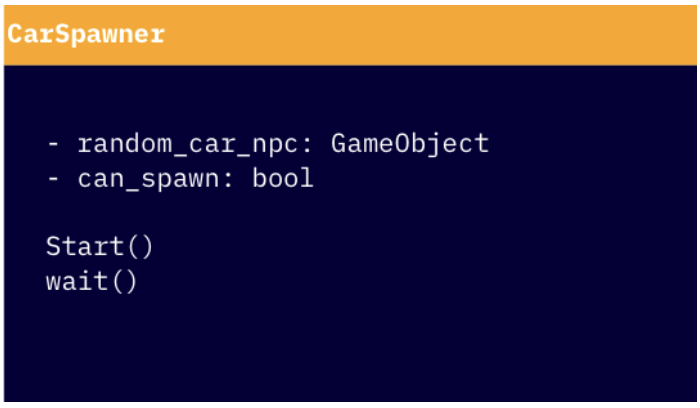


Figure 7: The Methods of CarSpawner Class

2.1.4. ScoreManager Class

This class starts with the required import statements. The first line imports the "System.Collections" namespace, which provides various collection classes. The second line imports the "System.Collections.Generic" namespace, which contains generic collection classes such as lists and dictionaries. The third line imports the "UnityEngine" namespace containing Unity engine functions. And finally the fourth line imports the "UnityEngine.UI" namespace, which allows interaction with the Unity UI system. The script then defines a class called "ScoreManager" which inherits from "MonoBehaviour". "MonoBehaviour" is the base class for Unity scripts and provides various methods for handling events and updates in the game.

There are two global variables inside the "ScoreManager" class. The first is "score_tx", which is of type "Text". This variable is used to refer to a UI Text component on the stage that will display the score. The second variable is the "score" variable, which is declared as a static floating point. The "Start" method is called once when the script is started. In this method, the `score` variable is set to 0 and the score is reset at the beginning of the game. The "Update" method is called in each frame of the game. Inside this method, the `text` property of the `score_tx` Text component is set to the `score` value, which is converted to a string using the `ToString()` method. This updates the score display in the UI with the current value of the "score" variable.

In summary, this script manages the score in the Unity game by updating a UI Text component with the current point value stored in the 'score' variable. The score can be increased or changed in other parts of the game, and this script allows the UI to reflect the updated score in real time.

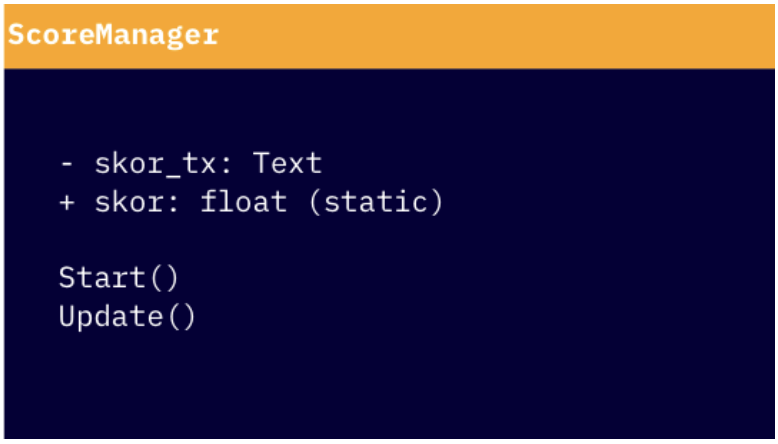


Figure 8: The Methods of ScoreManager Class

2.1.6. PlayButton Class

The PlayButton class is defined, which inherits from MonoBehaviour. MonoBehaviour is a base class for Unity scripts that allows them to interact with the Unity engine's runtime behavior.

Within the PlayButton class, there is a public method called "play()". This method is marked as public, meaning it can be accessed from other scripts or components. When this method is called, it executes the code inside its body. Inside the play() method, the SceneManager.LoadScene() function is called. This function is used to load a new scene based on its index. In this case, the index provided is 1, indicating the second scene in the build settings. When SceneManager.LoadScene(1) is called, Unity will load the scene with the corresponding index, and the current scene will be replaced by the new one.

Overall, this script sets up a PlayButton component that, when clicked, triggers a scene change to the second scene in the build settings, providing a basic functionality for a play button in a Unity game.

2.1.7. RestartButton Class

This script is responsible for handling the behavior of a button that restarts the game or reloads the initial scene when clicked.

In the first paragraph, similar to the previous script, this code has the required namespaces. "System.Collections", "System.Collections.Generic", "UnityEngine" and "UnityEngine.SceneManagement." Again, the RestartButton class is defined, which inherits from MonoBehaviour. Within this class, there is a public method called "crash()". This method is marked as public, indicating it can be accessed from other scripts or components. When this method is called, it executes the code inside its body.

Inside the crash() method, the SceneManager.LoadScene() function is called. This function is used to load a scene based on its index. In this case, the index provided is 0, indicating the first scene in the build settings. When SceneManager.LoadScene(0) is called, Unity will load the scene with the corresponding index, effectively restarting the game or reloading the initial scene.

Overall, this script sets up a RestartButton component that, when clicked, triggers a scene change to the first scene in the build settings, providing a functionality to restart the game. This script differs from the previous PlayButton script in terms of the scene index used for loading and the name of the method called when the button is clicked.

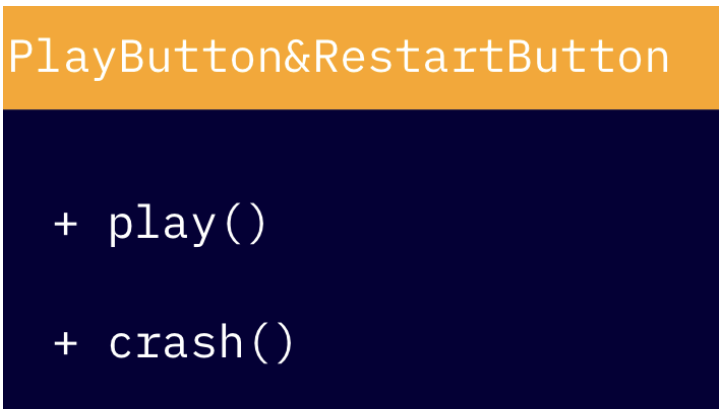


Figure 9: The Methods of PlayButton&Restart Button Class

2.2) Graphics

The game contains 2D graphics suitable for the highway and lane road atmosphere of the game. Graphs are shown in Figure 10-11.

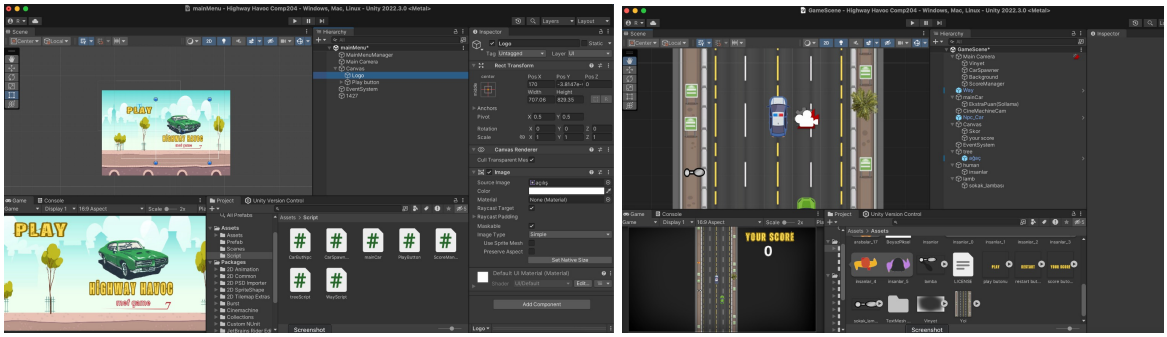
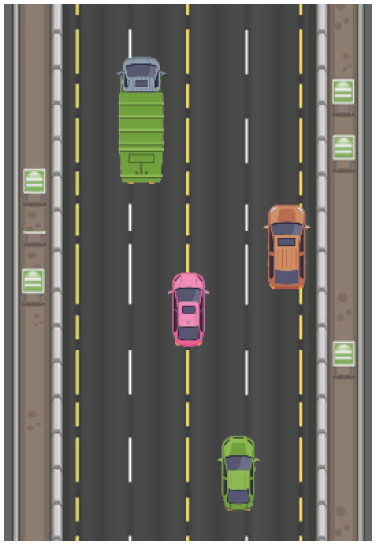


Figure 10-11: The Graphs of The Game

2.2.1. Lanes and Road System

In the game, cars are added randomly to the road for each lane in the game. The lanes are constantly repeated at road ends and the player has to decide which lane to cross based on the vehicle in front of them. The strip system is shown in Figure 12-14.



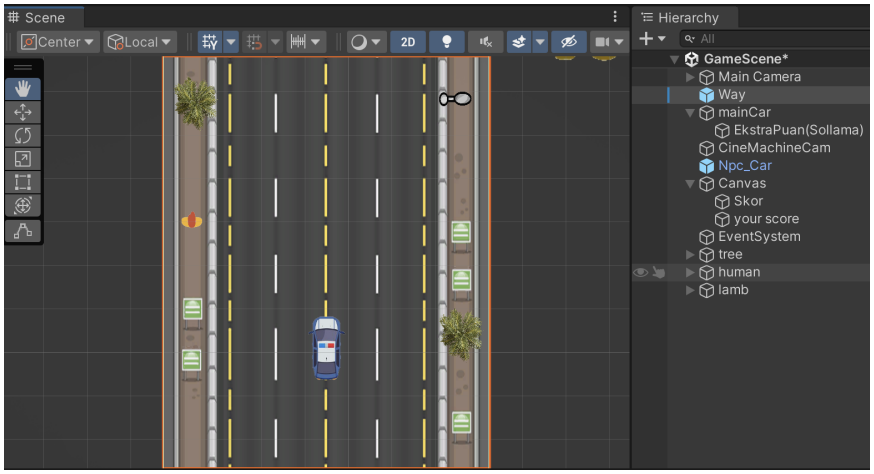


Figure 12-13-14: The Sprits System of The Way

2.2.2.NPC Cars System

The game also includes obstacle objects just like any other base game. The obstacles in this game are NPC cars. There are 10 types of cars we have defined. The boundaries of the player car are marked from the mirrors. If he touches other vehicles more than the mirror line, the player loses the game. NPC cars have a fixed speed. The obstacle is controlled by the CarSpawner and CarButNpc classes.

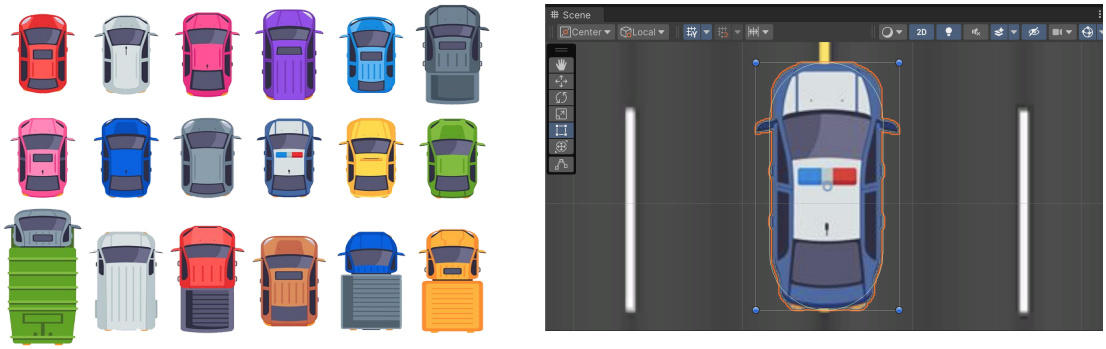


Figure 15-16: NPC Cars

2.2.3.Overtaking and Acceleration System

The player's speed is initially fixed and cannot move backwards. To accelerate, the forward arrow key must be pressed. If he moves too close to the other vehicles, his score is increased by 30 points instead of 10 points. This is an advantage for the player score.

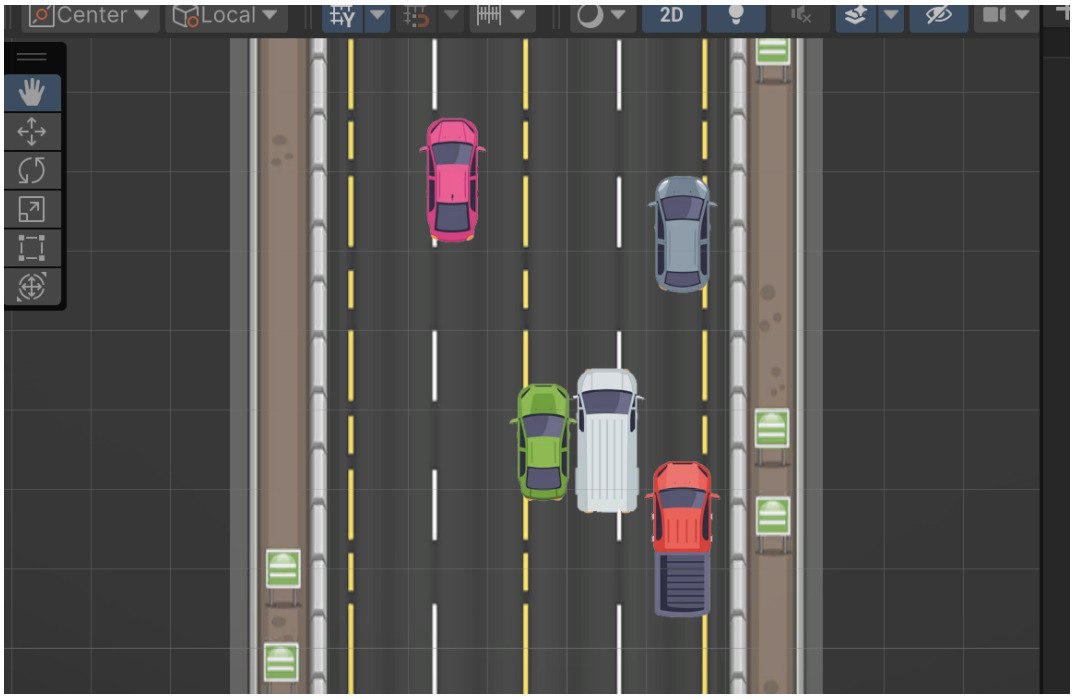


Figure 17: Overtaking and Acceleration System

2.2.4.Accident Mechanics

The game ends when the player hits one of the other cars, that is, when their borders touch one of the NPC cars. In this case, the game screen with the restart option appears, as shown in Figure 27.

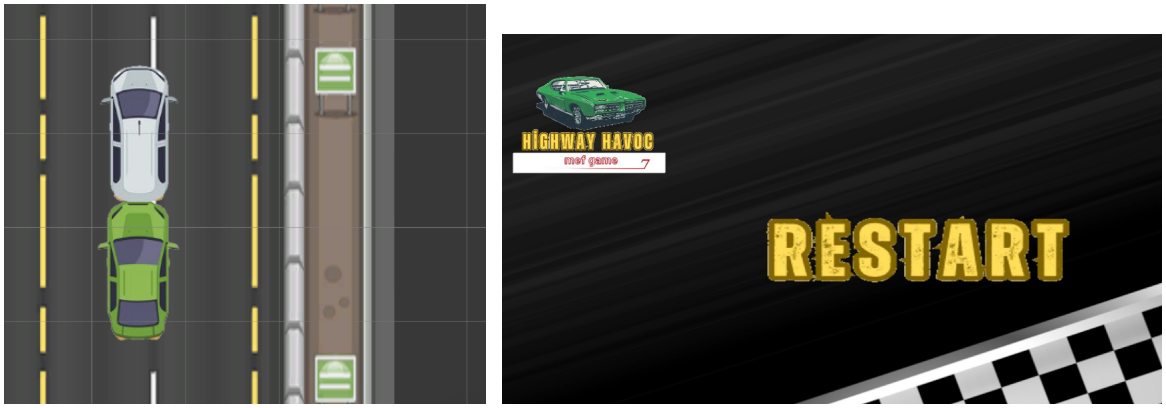


Figure 18-19: Accident Screen

2.2.5.Score

The game has been established with a score system in accordance with the game technique. Normally, the score increases by 10 depending on the elapsed time. In case of overtaking, this score is 30.

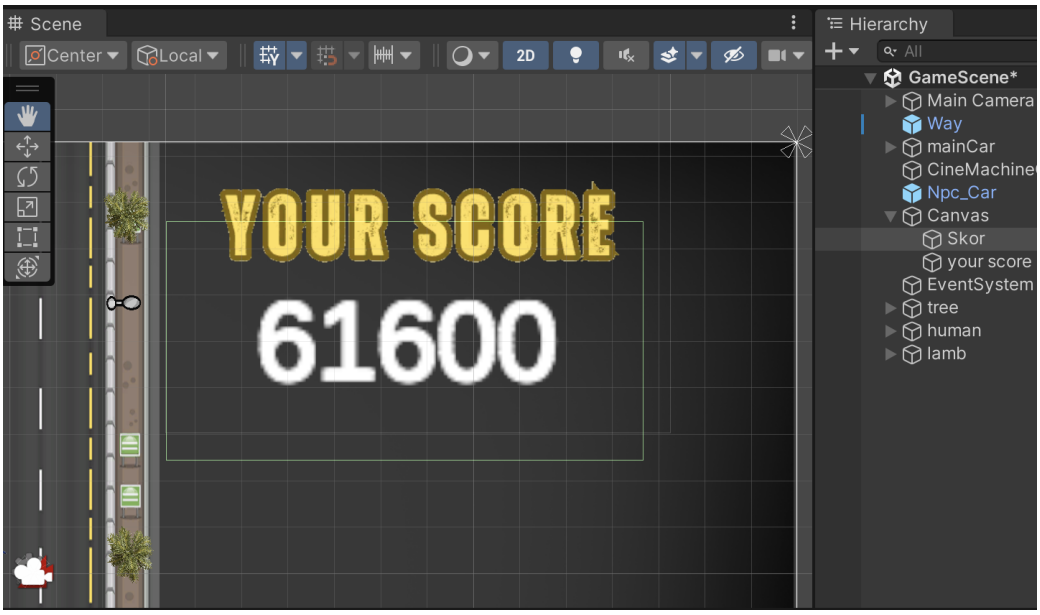


Figure 20: "Your Score" Screen

3) Achievements of the Project

As a result of the project, we successfully completed all the tasks assigned to us in the Blackboard system, exceeding our initial goals and expectations. This project took our first step into Unity game development and taught us the intricacies of creating immersive gaming experiences. Throughout the process, we gained experience about game development. We developed our skills in areas such as coding, level design and object creation.

Collaborating as a team on this project has benefited us for business life. We discovered the importance of effective communication, coordination and task delegation in a game development team. Working together we were able to overcome challenges, share ideas and optimize our workflow, ultimately improving our ability to work efficiently and cohesively as a unit. This project was a stepping stone in our coding journey to develop games. It has given us a solid foundation and confidence to undertake more ambitious projects in the future.

References

1- GDTitans. (2022). INSTALL & SETUP UNITY 🎮 | Getting Started [2023 Guide] [YouTube Video]. In *YouTube*. https://www.youtube.com/watch?v=gxX7euQ_2Qc

2- KaiJPR. (2022). How to make a Simple Car Game using Unity 2022! EP1 [YouTube Video]. In *YouTube*. https://www.youtube.com/watch?v=aKieQkknjm8

3-*Newest "unity-game-engine" Questions*. (2023). Stack Overflow. https://stackoverflow.com/questions/tagged/unity-game-engine?tab=Newest

4-peng, shelly. (2021, January 10). *Unity & C# - car going a little bit unsmooth on ramp*. Stack Overflow. https://stackoverflow.com/questions/65653017/unity-c-sharp-car-going-a-little-bit-unsmooth-on-ramp

5-*Unity Real-Time Development Platform | 3D, 2D, VR & AR Engine*. (2022). Unity.com. https://unity.com

6-Wikipedia Contributors. (2023, May 31). *Unity (game engine)*. Wikipedia; Wikimedia Foundation. https://en.wikipedia.org/wiki/Unity_(game_engine)