Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка

до кваліфікаційної роботи
бакалавра

на тему: «Розробка мобільного 3D-додатку X-Racer. Модулі механіки.
за освітньою програмою «12 Інженерія програмного забезпечення»
зі спеціальності: «121 Інженерія програмного забезпечення»
Виконав: студент групи «ПЗ1912»

_____ /Дмитро ПЯНИЙ/
(підпис студента) (Ім'я ПРІЗВИЩЕ)

Керівник: _____ /доц. Вадим АНДРЮЩЕНКО
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Нормоконтролер: _____ /Світлана ВОЛКОВА/
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Засвідчую, що у цій роботі немає запозичень з
праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»
Department «Computer information technology»

Explanatory Note
to Bachelor's Thesis

on the topic: «Development of mobile 3d application X-Racer. Mechanics modules»
according to educational curriculum « Software engineering »
in the Speciality: «121 Software engineering»

Done by the student of the group PZ1912: / /

Scientific Supervisor: /Vadym ANDRIUSHCHENKO/

Normative controller: / /

Dnipro – 2023

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: «Комп'ютерні технології і системи»
Кафедра: «Комп'ютерні інформаційні технології»
Рівень вищої освіти: бакалавр
Освітня програма: «Інженерія програмного забезпечення»
Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
_____/Вадим ГОРЯЧКІН/
(підпис)
Дата_____

З А В Д А Н Н Я

На кваліфікаційну роботу
бакалавра студенту П`яному Дмитру Анатолійовичу
1. Тема роботи: «Розробка мобільного 3D-додатку X-Racer. Модулі механіки»
Керівник роботи: Андрющенко Вадим Олександрович, доцент
затверджені наказом № 445 ст від 16.05.2023.
2. Строк подання студентом роботи: 15.06.2023
3. Вихідні дані до роботи:
4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати):
вступ, проєктування, реалізація головних механік 3D-додатку, тестування, висновки, література.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): презентація, відео-демонстрація роботи програми.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів	Примітка
1	Вступ	03.04.2023 – 05.04.2023	
2	Огляд літератури Unity, фізика 3D-об`єктів	06.04.2023 – 09.04.2023	
3	Проєктування основних модулів механіки	10.04.2023 – 24.04.2023	
4	Постановка завдання	25.04.2023 – 02.05.2023	
5	Розробка та програмування логіки механік	03.05.2023 – 10.05.2023	30%
6	Розробка та програмування додаткового функціоналу	11.05.2023 – 18.05.2023	
7	Тестування	19.05.2023 – 26.05.2023	60%
8	Оформлення пояснювальної записки	27.05.2023 – 02.06.2023	
9	Розробка демонстраційних матеріалів	03.06.2023 – 18.06.2023	100%
10	Подання кваліфікаційної роботи до кафедри	19.06.2023 – 26.06.2023	
11	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	27.06.2023	

Студент _____ /Дмитро П`ЯНИЙ/

Керівник роботи _____/Вадим АНДРЮЩЕНКО/

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра: 149 с., 34 рис., 2 табл., 4 додатки, 12 джерел.

Об’єкт розробки – основні механіки додатку.

Мета роботи - розробка різних методів управління автомобілем, реалістичної моделі фізики руху автомобіля, штучного інтелекту ботів, створення системи винагород та вдосконалення автомобілів гравця, включаючи модифікацію та покращення різних аспектів автомобіля.

Ключові слова: UNITY; ТЕСТ КЕЙСИ; СКРИПТ; ASSETS; GAMEOBJECTS;

ЗМІСТ

ВСТУП

У сучасному світі мобільні додатки стали невід'ємною частиною нашої повсякденної життя. Вони дозволяють нам здійснювати різноманітні дії та задовольняти наші потреби у будь-яку годину та в будь-якому місці. Водночас зростає інтерес до використання технологій 3D-моделювання та віртуальної реальності для створення захоплюючих іммерсивних досвідів для користувачів.

Метою роботи була розробка мобільного додатку, що надає гравцям можливість поринути у захоплюючий світ гонок на високій швидкості та випробувати адреналін, що супроводжує цей вид спорту. Основними завданнями проекту були створення реалістичного ігрового оточення, реалізація захоплюючого геймплею, розробка системи керування автомобілем, а також додавання елементів прогресії.

Для досягнення поставлених цілей та завдань я використав сучасні інструменти та технології розробки мобільних додатків. У процесі роботи були використана мова програмування C#, ігровий двигун Unity, 3D-моделювання та фізична симуляція.

Результатом цієї роботи стало створення мобільного 3D-додатку X-Racer, яке пропонує гравцям захоплюючий геймплей, можливість покращення автомобіля, різноманітність рівнів та багато іншого.

РОЗДІЛ 1
ОГЛЯД СЕРЕДОВИЩА РОЗРОБКИ UNITY.

1.1 Основні поняття

Unity - це потужне інтегроване середовище розробки (IDE) для створення ігор та інтерактивних додатків. Воно надає широкий набір інструментів і можливостей для розробки, від створення графіки та анімації до програмування та тестування. Простота використання Unity робить виробництво ігор максимально простим та зручним, а мультиплатформенність дозволяє розробникам ігор працювати на максимально можливій кількості ігрових платформ та операційних систем.

Редактор має досить простий Drag&Drop інтерфейс, який складається з різних вікон, завдяки чому можна проводити налагодження гри прямо в редакторі. Движок підтримує дві скриптові мови: C#, JavaScript (модифікація). Розрахунки фізики здійснює фізичний движок PhysX від NVIDIA.

Проект в Unity ділиться на сцени (рівні) - окремі файли, що містять свої ігрові світи зі своїм набором об'єктів, сценаріїв, і налаштувань. Сцени можуть містити в собі як, власне, об'єкти (моделі), так і порожні ігрові об'єкти - об'єкти, які не мають моделі. Об'єкти, в свою чергу містять набори компонентів, з якими і взаємодіють скрипти.

У будь-якого об'єкта на сцені обов'язково присутній компонент Transform - він зберігає в собі координати місця розташування, повороту і розмірів об'єкта по всіх трьох осях. У об'єктів з видимою геометрією також за замовчуванням присутній компонент Mesh Renderer, що робить модель об'єкта видимою.

Кожен Unity - проект містить папку Assets. Вміст цієї папки подано в Project View. Це ресурси гри: скрипти, 3D-моделі, текстури, аудіофайли, префаби (Prefabs).

Деякі ресурси створюються безпосередньо в Unity. Панель Hierarchy містить всі об'єкти (GameObject) відкритої сцени. Деякі з них - це екземпляри ресурсів (3D-моделі та інше), інші - екземпляри префабів. У Hierarchy об'єктів можна задавати успадкування. Об'єкти, що додаються в сцену або видаляються з неї, відображаються або навпаки перестають відображатися в Hierarchy (мал.1).

Рисунок 1.1 Панель Hierarchy

Unity використовує концепцію успадкування (Parenting). Будь-який об'єкт може бути дочірнім по відношенню до іншого. Для цього достатньо перетягнути його на «батька» в Hierarchy. Дочірній об'єкт буде рухатися і обертатися разом з батьком (рис. 1.2).

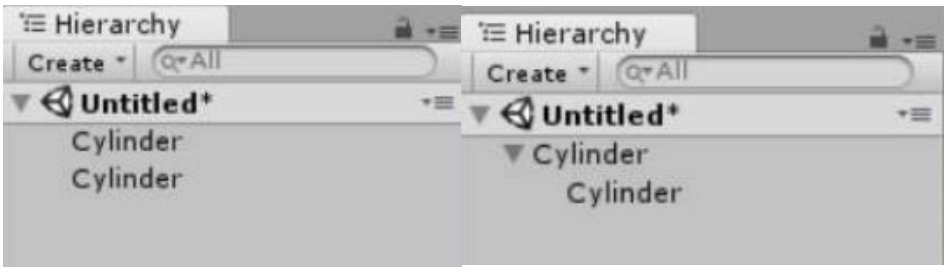


Рисунок 1.2 Панель Hierarchy: а) два незалежні об'єкти; б) один об'єкт є дочірнім

1.2 Складові середовища

- Сцени та об'єкти

Unity використовує концепцію сцен та об'єктів. Сцена - це контейнер, у якому розміщуються об'єкти та інші ресурси. Об'єкти являють собою ігрові об'єкти, такі як персонажі, перешкоди, камери, світло та інші елементи сцени.

- Компоненти

Об'єкти в Unity містять компоненти, які визначають їхню поведінку та функціональність. Компоненти можуть бути скриптами, які пишуться мовою C# або JavaScript, або встановленими компонентами, такими як компоненти керування анімацією, фізики, графіки та звуку.

- Скрипти

Розробники можуть створювати скрипти для керування поведінкою об'єктів та реалізації ігрової логіки. Unity підтримує скрипти кількома мовами програмування, але найпоширенішим є C#.

- Графічний редактор

Unity надає графічний редактор, який дозволяє розробникам створювати та редагувати сцени, об'єкти, матеріали та інші ресурси. Редактор має інтуїтивно зрозумілий інтерфейс та надає інструменти для візуального розміщення об'єктів, налаштування властивостей та створення простих анімацій.

- Фізика

Unity включає вбудований движок фізики, який обробляє зіткнення, гравітацію та інші фізичні ефекти. Розробники можуть застосовувати фізичні властивості до об'єктів і контролювати їхню поведінку.

1.3 Фізика 3D – об`єктів. Її компоненти

- Rigidbodies (Тверде тіло)

Для того, щоб підпорядкувати об'єкт законам фізики в Unity 3D, необхідно додати до цього об'єкта компонент Rigidbody. Тоді на об'єкт буде діяти гравітація і він буде стикатися з іншими об'єктами.

Компонент Rigidbody має властивість «Is Kinematic», яка дозволяє вилучити об'єкт з-під контролю фізичного движка, і дозволити переміщати його кінематично за допомогою скрипта. Значення «Is Kinematic» можна змінювати за допомогою коду, щоб ввімкнути або вимкнути фізику для об'єкта, але ця можливість вимагає додаткових ресурсів.

- Colliders (Колайдери)

Колайдери це наступний тип компонентів, які повинні бути додані до твердих тіл, щоб задіяти зіткнення. Якщо два твердих тіла врізаються один в одного, фізичний движок не буде прораховувати зіткнення, поки до обох об'єктів не буде додано колайдер. Тверді тіла, які не мають колайдерів будуть просто проходити крізь один одного при прорахунку зіткнень.

- Статичний колайдер (Static Collider)

Це ігровий об'єкт, у якому є колайдер, але не Rigidbody. Статичні колайдери використовуються для геометричних рівнів, які завжди стояти на місці і зовсім не розвиваються. Зовнішні об'єкти Rigidbody будуть врізані в статичний колайдер, але його не зрушити.

Отже, статичні колайдери не можна вмикати/ вимикати, рухати або масштабувати під час ігрового процесу. Якщо ви зміните статичний колайдер, то в результаті фізичним движком буде викликаний додатковий внутрішній

перерахунок, який буде супроводжуватися падінням продуктивності. Гірше того, зміни іноді можуть залишити колайдер в невизначеному стані, в результаті чого будуть проводитися помилкові фізичні розрахунки. З цих причин, слід змінювати тільки колайдери з Rigidbody.

- Rigidbody колайдер (Rigidbody Collider)

Це ігровий об'єкт, до якого прикріплений колайдер і некінематичний Rigidbody. Rigidbody колайдери повністю симулюються фізичним движком і можуть реагувати на колізії і сили, прикладені з скрипта. Вони можуть стикатися з іншими об'єктами (включаючи статичні колайдери) і є найпоширенішою конфігурацією колайдера в іграх, які використовують фізику

- Кінематичні Rigidbody колайдери (Kinematic Rigidbody Collider)

Це ігровий об'єкт, до якого прикріплений колайдер і кінематичний Rigidbody (тобто властивість IsKinematic компонента Rigidbody включено). Змінюючи компонент Transform, ви можете переміщати об'єкт з кінематичним Rigidbody, але він не буде реагувати на колізії та додані сили так само, як і некінематичні Rigidbody. Кінематичні Rigidbody повинні використовуватися для колайдерів, які можуть рухатися або періодично вимикатися/вмикатися, інакше вони будуть вести себе як статичні колайдери.

На відміну від статичного колайдера, кінематичний Rigidbody буде застосовувати тертя до інших об'єктів і, в разі контакту, буде «будити» інші Rigidbody. Навіть коли вони нерухомі, кінематичні Rigidbody колайдери поводяться інакше, на відміну від статичних колайдерів.

1.4 Unity фреймворки

Фреймворк – це програмне середовище, яке спрощує та прискорює створення програмного забезпечення. За використання фреймворків ви пишете лише код, який реалізує логіку, специфічну для вашого продукту. Вам не доводиться самостійно забезпечувати роботу з базою даних, автентифікацію, підтримку сеансів тощо. Все це реалізовано у фреймворках.

Фреймворк визначає архітектуру програми та забезпечує взаємодію між її компонентами. Він містить різні бібліотеки та використовує їх для створення каркасу програми. Коли ви працюєте з фреймворком, відбувається інверсія управління. Не користувач, а фреймворк встановлює, коли викликати функції користувача.

1.4.1 Класифікація фреймворків в Unity

Unity Ads: фреймворк, призначений для інтеграції реклами гри. Він дозволяє розробниками монетизувати свої проекти, включаючи вбудовану рекламу, відеорекламу та інші формати.

Unity AI: фреймворк для розробки та впровадження штучного інтелекту в додатки. Пропонує набір з інструментів та бібліотек для створення розумних агентів, систем поведінки та алгоритмів машинного навчання.

Unity UI: фреймворк для створення інтерфейсу користувача в додатках. Він представляє з себе такі компоненти як кнопки, текстові поля, панелі та інші елементи інтерфейсу, а також інструменти для стилізації і взаємодії з користувачем.

Unity Cinemachine: фреймворк для створення кінематографічної камери та управління камерними сценами. Він дозволяє розробникам створювати складні

камерні плани, налаштовувати положення, композицію, переміщення та переходи між камерами. Cinemachine полегшує створення якісної та динамічної кінематографії в іграх та додатках.

Unity Timeline: фреймворк для створення та управління тимчасовою лінією подій в іграх та додатках. Він дозволяє розробникам створювати анімації, сценарії та події, налаштовувати їх часові параметри та взаємодію з іншими системами. Timeline дозволяє створювати складні сцени та послідовності дій без необхідності писати великий обсяг коду.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ОСНОВНИХ МОДУЛІВ 3D-ДОДАТКУ X-RACER

2.1 Структура програми. Механіка рівнів

Спочатку потрібно чітко визначити, які цілі потрібно досягти в грі і яка загальна концепція має бути реалізована. Наприклад, цілі можуть включати проходження рівнів, здобуття певної кількості очок або виконання певних завдань. Механіка підвищення рівня персонажа використовується в тому числі заради того, щоб гравець мав мотивацію грати далі.

Реалізація поставленого завдання:

У мобільній 3D грі "X-Racer" рівні є різними ігровими сценами або завданнями, які гравець проходить у міру просування в грі. Кожен рівень є певною гоночною трасою, в якому гравцеві необхідно керувати своїм гоночним автомобілем і досягти певних цілей або виконувати завдання. Цілі рівнів можуть бути різними, включаючи збирання чек-поїнтів, обгін автомобіля, їзда без використання гальмів та багато інших.

Кожен рівень поділений на різні сегменти чи етапи, які гравець має пройти послідовно. Проходження рівнів упроводжується набором правил, таких як обмеження швидкості, контроль поведінки автомобіля, система балів або та можливості для покращення автомобіля.

Рівні в грі "X-Racer" мають різну складність і послідовність, надаючи гравцеві виклик, що поступово збільшується, і нові можливості в процесі гри. Як гравець просувається у грі та успішно завершує рівні, йому доступні нові рівні, траси, транспортні засоби чи інші можливості для розширення ігрового досвіду.

2.1.1 Кількість розроблених рівнів

Постановка завдання: проектування рівнів структур, розробка механіки рівнів та балансування складності.

Для початку нам необхідно розробити загальну структуру рівнів, яка включає послідовність, складність та прогресію рівнів. Визначимо, скільки рівнів буде в грі, як вони будуть пов’язані між собою та як вони змінюватимуться певною складністю або іншими параметрами.

Потім після визначення загальної структури рівнів потрібно розробити конкретні механіки, які будуть присутні у кожному рівні. Це можуть бути, завдання, вороги, перешкоди чи інші елементи, які гравець повинен буде вирішити чи подолати для успішного завершення рівня.

Щодо рівнів, так вони повинні бути відповідні для гравців різного рівня навичок. При проектуванні важливо збалансувати складність рівнів таким чином, щоби гравці відчували себе викликаними, але не пригніченими великою складністю на початку. Тому у грі повинні бути присутні різні рівні складності, щоб гравці мали змогу поступово розвиватися.

Після визначення складових функціоналу, відповідно до постановки розробляється задана кількість рівнів, їх основна мета та складність.

Реалізація поставленого завдання:

Level 1 Point Digger

Рівень "Point Digger" у грі є завданням, в якому гравцеві потрібно зібрати певну кількість чекпоінтів. Гравець повинен використовувати свої навички

керування автомобілем, щоб максимізувати свій рахунок та набрати необхідну кількість чекпоінтів.

Level 2 Over Take

Рівень являє собою ситуацію, в якій гравцеві потрібно переслідувати червоний автомобіль, яка є основним об'єктом на цьому етапі.

Level 3 No brake

На цьому рівні гравцеві заборонено використовувати гальма у своєму автомобілі. Це створює додаткові труднощі та вимагає від гравця особливих навичок та стратегії.

На рівні "No Brake" гравець повинен керувати автомобілем та досягати різних цілей, не використовуючи гальма. Це означає, що гравець повинен повністю покладатися на свої навички керування, реакцію та здатність передбачати рух інших автомобілів та перешкод на дорозі.

Level 4 Death Racer

На рівні "Death Racer" гравцеві надається можливість підібрати покращення, яке перетворює його автомобіль на вантажівку. У такому вигляді гравець повинен збити чотири автомобілі, що знаходяться на його шляху. Це створює додаткову складність, тому що вантажівка, як правило, має більшу масу і повільнішу швидкість, ніж звичайні автомобілі.

Гравцеві може знадобитися використовувати свої навички керування, щоб утримувати вантажівку на трасі, уникати перешкод і одночасно прагнути збити цільові автомобілі. Можливо, на рівні є різні перешкоди чи інші автомобілі, які заважають гравцеві досягти своєї мети.

Ігрові рівні з такими завданнями вимагають від гравця точності, реакції та стратегічного мислення. Поступове збивання всіх чотирьох автомобілів та досягнення мети є основними завданнями на цьому рівні.

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

public class ShieldTruck : MonoBehaviour
{
    public GameObject[] deactivator;
    void OnEnable()
    {
        foreach (GameObject d in deactivator) {
            d.SetActive (false);
        }
    }
}
```

Рисунок 2.1 Скрипт перетворення автомобіля на вантажівку

Level 5 Point Digger

Рівень "Point Digger" у грі являє собою завдання, в якому гравцеві потрібно зібрати певну кількість чекпоїнтів. Щоб набирати очки, гравець має успішно виконувати різні дії у грі, такі як обганяти інші автомобілі, здійснювати близькі пропускання тощо. Кожне успішне виконання таких дій принесе гравцю поїнти.

Level 6 Police Chase

Рівень "Police Chase" у грі є завданням, в якому гравцю необхідно втекти від поліцейської погоні, перш ніж його зловлять.

Мета цього рівня полягає в тому, щоб проїхати певну відстань до того, як поліцейська машина зможе наздогнати гравця. Відстань, яку потрібно подолати, вказано в ігровій інформації рівня. Під час погоні гравець має бути обережним і уникати зіткнень із поліцейськими автомобілями, щоб не втратити швидкість та не бути затриманим.

Якщо гравець зможе подолати задану відстань до того, як його зловлять поліцейські, рівень буде успішно завершено. В іншому випадку, якщо гравця буде затримано поліцією, рівень буде провалено.

Рівень "Police Chase" представляє динамічну та захоплюючу ігрову ситуацію, де гравець має проявити навички управління та вміння уникати переслідування поліції.

Level 7 Risk

Рівень "12 Time Near Miss" - це рівень, який вимагає від гравця майстерності в керуванні автомобілем та точного проходження повз інші автомобілі. Головне завдання на цьому рівні полягає в тому, щоб успішно проскочити повз 12 автомобілів, минаючи їх із мінімальним зазором. На рівні "12 Time Near Miss" гравець має бути дуже уважним і реагувати швидко, щоб уникати зіткнень з іншими автомобілями. Гравцю надається обмежена кількість спроб, щоб проскочити повз всі 12 автомобілів, і зіткнення з будь-яким з них може призвести до невдачі на рівні.

Level 8 Overspeed

Є викликом, в якому гравцеві необхідно набрати 20000 очок у заданий проміжок часу 90 секунд. Поїнти зазвичай набираються шляхом виконання певних дій, таких як подолання перешкод, проходження через ворота, збирання спеціальних предметів на трасі. Чим складніші дії робить гравець, тим більше очок він отримує.

Level 9 Point Digger

Рівень "Point Digger Check Point Collect" є завданням, в якому гравцеві необхідно зібрати сім контрольних точок на трасі. Контрольні точки можуть бути позначені певними об`єктами чи символом на трасі.

Мета гравця полягає в тому, щоб успішно пройти через кожну контрольну точку, слідуючи певному маршруту. Після проходження кожної контрольної точки, гравець може отримати поїнти, які можуть впливати на

загальний результат. Проходження рівня може представляти певні виклики та перешкоди, такі як перешкоди на трасі.

Level 10 Wrong Drive

Є викликом, в якому гравцеві потрібно керувати автомобілем, рухаючись у неправильному напрямку протягом 20 секунд. Мета рівня полягає в тому, щоб утримувати свій автомобіль у неправильному напрямку якомога довше, долаючи перешкоди та уникаючи зіткнень з іншими автомобілями. При цьому гравець може стикатися з різними викликами, такими як вузькі ділянки дороги, різкі повороти або перешкоди на шляху.

Level 11 Extreme Drive

Рівень Shield Collect, Destroy Vehicles 3 являє собою завдання, в якому гравцеві необхідно підібрати щит та знищувати автомобілі. Головна мета рівня полягає в тому, щоб набрати певну кількість очок, знищуючи автомобілі, використовуючи підібраний щит.

Під час проходження рівня гравець повинен акуратно керувати своїм автомобілем та активно шукати щити, які можуть з`являтися на трасі. Зібрані щити перетворюють автомобіль користувача на вантажівку, дозволяючи гравцеві врізатися в інші автомобілі для їх знищення.

Level 12 Risk

Рівень, в якому користувачу потрібно встигнути проїхати повз 10 автомобіля за певний час 90 секунд.

Level 13 Rush Drive

Рівень "Rush Drive: Wrong Way Drive – 8000 Score" є завданням, в якому гравцеві потрібно набрати 8000 поінтів, рухаючи у напрямку, протилежному дозволеному движенню (Wrong Way Drive). У цьому рівні важливо, не лише зібрати необхідну кількість очок, але й упоратися з керуванням автомобіля, рухаючись у протилежному напрямку щодо звичайного руху. Вам потрібно акуратно керувати автомобілем, щоб уникати зіткнень та одночасно збирати поїнти.

Level 14 Speed Bonus

Є завданням, в якому гравцеві потрібно набрати 30000 поїнтів, не використовуючи гальма. У цьому рівні важливо підримувати високу швидкість та збирати поїнти, мінімізуючи використання гальм.

Level 15 Risk

Є викликом, в якому гравцеві необхідно провскочити повз 10 поліцейських автомобілів, минаючи їх із мінімальним зазором та при цьому не даючи вас арештувати.

Нижче представлена програмна побудова рівнів та їх послідовність (рис. 2.2):

```
void Start()
{
    levelInfoParent.SetActive (true);
    totalPuzzleCharacters = 0;
    if (_level == 1) {
        levelInfo.text = "Collect Check Point: 4";//+ CollectableControl.CountCollect
    }
    if (_level == 11) {  //shield collection
        levelInfo.text = "Collect shield";
        destroyedVehicles = 3;
        // Change
    } else if (_level == 3) { //drive without brake
        levelDistance = 2;
        MobileController._mobile.brake.gameObject.SetActive (false);
        levelInfo.text = "Drive 2KM without brake";
    } else if (_level == 10) { //wrong way driving
        levelTime = 20f;
        levelInfo.text = "Wrong way driving for " + levelTime + " s";
        PlayerPrefs.SetInt ("SelectedModeIndex", 1); //mode must be two way
        HR_GamePlayHandler.Instance.reSetTrafficMode ();
    }
    else if (_level == 5) {
        levelInfo.text = "Collect Check Point:5";
```

Рисунок 2.2 Програмна послідовність рівнів

Цей код визначає поведінку функції Start() на основі значення змінної _level. Розглянемо кожну умову:

Якщо _level дорівнює 1:

- Об'єкт levelInfoParent стає активним.
- Змінна totalPuzzleCharacters встановлюється в 0.
- Текст levelInfo встановлюється в "Зберіть контрольну точку: 4".

Якщо _level дорівнює 11:

- Об'єкт levelInfoParent стає активним.
- Текст levelInfo встановлюється в "Зберіть щит".
- Змінна destroyedVehicles встановлюється в 3 рази.

Якщо _level дорівнює 3:

- Значення змінної рівняDistance встановлюється в 2.

- Об'єкт MobileController._mobile.brake.gameObject стає неактивним (вимикається).
- Текст levelInfo встановлюється у "Проїхати 2 км без використання гальма".

Якщо _level дорівнює 10:

- Значення змінної levelTime встановлюється 20.
- Текст levelInfo встановлюється в "Рух зустрічним шляхом протягом + levelTime + секунд".
- Викликається функція PlayerPrefs.SetInt("SelectedModeIndex", 1), яка встановлює значення "SelectedModeIndex" в 1.
- Викликається функція HR_GamePlayHandler.Instance.reSetTrafficMode(), яка скидає режим руху.

Якщо _level дорівнює 5:

- Текст levelInfo встановлюється в "Зберіть контрольну точку: 5".
- Кожна умова визначає різні дії, які виконуються залежно від _level.

Продовження програмної структури (рис. 2.3):

```
}
else if ( _level == 9)
{
    levelInfo.text = "Collect Check Point:7";
}

else if (_level == 6) {
    levelDistance = 2;
    levelInfo.text = "Drive " + levelDistance + " kM before police chase you";
}
else if (_level == 7) { //near miss
    levelNearMiss = 12;
    levelInfo.text = "Collect " + levelNearMiss + " Near Miss";
}
else if (_level == 8) { //collect scores
    levelTime = 90f;
    levelScore = 20000f;
    levelInfo.text = "Collect " + levelScore + " score in " + levelTime + " s";
}
else if (_level == 4) {
    destroyedVehicles = 4;
    levelInfo.text = "Collect Booster Shield";
}
```

Рисунок 2.3 – Послідовність рівнів

Якщо _level дорівнює 9:

- Текст levelInfo встановлюється в "Зберіть контрольну точку: 7".

Якщо _level дорівнює 6:

- Значення змінної рівняDistance встановлюється в 2.
- Текст levelInfo встановлюється в "Проїхати" + levelDistance + "км, перш ніж поліція почне переслідування".

Якщо _level дорівнює 7:

- Значення змінної рівнів невеликіймісці встановлюється в 12.
- Текст levelInfo встановлюється в "Зберіть" + levelNearMiss + "близьких перепусток повз інші автомобілі".

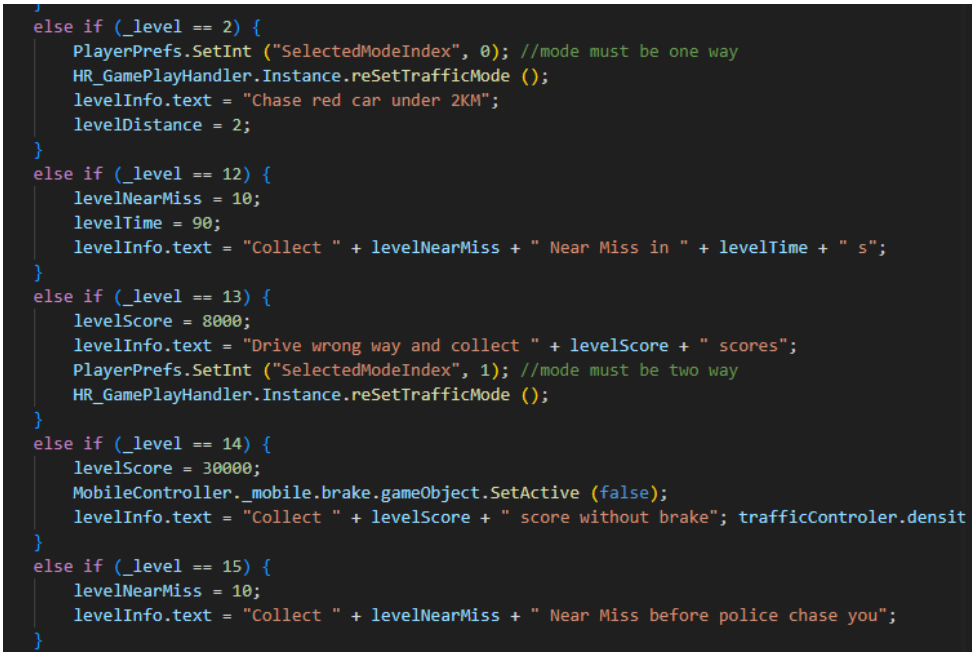
Якщо _level дорівнює 8:

- Значення змінної levelTime встановлюється 90.
- Значення змінної levelScore встановлюється 20000.
- Текст levelInfo встановлюється в "Зберіть " + levelScore + " очок за " + levelTime + " секунд ".

Якщо _level дорівнює 4:

- Значення змінної забрудненихвеглин встановлюється в 4 рази.
- Текст levelInfo встановлюється в "Зберіть захисний щит".

Продовження програмної структури (рис. 2.4)



```
else if (_level == 2) {
    PlayerPrefs.SetInt ("SelectedModeIndex", 0); //mode must be one way
    HR_GamePlayHandler.Instance.reSetTrafficMode ();
    levelInfo.text = "Chase red car under 2KM";
    levelDistance = 2;
}
else if (_level == 12) {
    levelNearMiss = 10;
    levelTime = 90;
    levelInfo.text = "Collect " + levelNearMiss + " Near Miss in " + levelTime + " s";
}
else if (_level == 13) {
    levelScore = 8000;
    levelInfo.text = "Drive wrong way and collect " + levelScore + " scores";
    PlayerPrefs.SetInt ("SelectedModeIndex", 1); //mode must be two way
    HR_GamePlayHandler.Instance.reSetTrafficMode ();
}
else if (_level == 14) {
    levelScore = 30000;
    MobileController._mobile.brake.gameObject.SetActive (false);
    levelInfo.text = "Collect " + levelScore + " score without brake"; trafficControler.densit
}
else if (_level == 15) {
    levelNearMiss = 10;
    levelInfo.text = "Collect " + levelNearMiss + " Near Miss before police chase you";
}
```

Рисунок 2.4 Послідовність рівнів

Якщо _level дорівнює 2:

- Викликається функція PlayerPrefs.SetInt("SelectedModeIndex", 0), яка встановлює значення "SelectedModeIndex" у 0.

- Викликається функція HR_GamePlayHandler.Instance.reSetTrafficMode(), яка скидає режим руху.
- Текст levelInfo встановлюється в "Переслідуйте червону машину протягом 2 км".
- Значення змінної рівняDistance встановлюється в 2.

Якщо _level дорівнює 12:

- Значення змінної рівнів невеликіймісці встановлюється в 10.
- Значення змінної levelTime встановлюється 90.
- Текст levelInfo встановлюється в "Зберіть" + levelNearMiss + "близьких перепусток повз інші автомобілі за " + levelTime + " секунд ".

Якщо _level дорівнює 13:

- Значення змінної levelScore встановлюється 8000.
- Текст levelInfo встановлюється в "Рухайте зустрічним шляхом і зберіть "+levelScore+" очок.
- Викликається функція PlayerPrefs.SetInt("SelectedModeIndex", 1), яка встановлює значення "SelectedModeIndex" в 1.
- Викликається функція HR_GamePlayHandler.Instance.reSetTrafficMode(), яка скидає режим руху.

Якщо _level дорівнює 14:

- Значення змінної levelScore встановлюється 30000.
- Об'єкт MobileController._mobile.brake.gameObject стає неактивним (вимикається).

- Текст levelInfo встановлюється в "Зберіть + levelScore + Score без використання гальма".
- Значення змінної trafficControler.density встановлюється 10.

Якщо _level дорівнює 15:

- Значення змінної рівнів невеликіймісці встановлюється в 10.
- Текст levelInfo встановлюється в "Зберіть" + levelNearMiss + "Near Miss повз інші автомобілі, перш ніж поліція почне переслідування".

2.1.2 Односторонній та двосторонній рух

Постановка завдання: створення другорядного функціоналу, основною задачею якого є додавання різноманітності до гри. Концепція вибору руху може додавати відчуття небезпечності та викликати адреналін.

Реалізація функціоналу:

Вибираючи трасу і ландшафт в грі, вам буде запропоновано вибрати односторонній або двосторонній рух. Це дозволить вам визначити, чи будете ви рухатися лише в одному напрямку на трасі, чи матимете можливість рухатися в обох напрямках.

Рисунок 2.5 Односторонній та двосторонній рух

Нижче представлена програмна реалізація одностороннього та двостороннього руху (рис. 2.6):

```
#region traffic mode selection methods
void initializeTrafficMode()
{
    //bestScoreOneWay.text = PlayerPrefs.GetInt("bestScoreOneWay").ToString();
    //bestScoreTwoWay.text = PlayerPrefs.GetInt("bestScoreTwoWay").ToString();
    //bestScoreTimeTrail.text = PlayerPrefs.GetInt("bestScoreTimeAttack").ToString();
}
public void selectFreeTrafficMode(int index) //one way:0, two way:1, time trail:2
{
    PlayerPrefs.SetInt("SelectedModeIndex", index);
    //topBar.SetActive(false);
    //descriptionBox.SetActive(true);
    //LevelNextButton.gameObject.SetActive(true);
    Application.LoadLevel("LoadingScreen");  //8
```

Рисунок 2.6 Вибір режиму руху

Цей код містить дві функції, які стосуються вибору режиму руху:

Функція initializeTrafficMode():

- У цій функції код використовувався для встановлення кращих результатів у режимах "односторонній рух", "двосторонній рух" та "гра на час". Код отримує значення з PlayerPrefs і надає їм відповідні змінні елементам інтерфейсу, таким як bestScoreOneWay, bestScoreTwoWay та bestScoreTimeTrail.

Функція selectFreeTrafficMode(int index):

- Ця функція приймає параметр index, який визначає вибраний режим руху: односторонній рух (значення 0), двосторонній рух (значення 1) або гра на час (значення 2).
- Потім значення index зберігається в PlayerPrefs за допомогою функції PlayerPrefs.SetInt("SelectedModeIndex", index).
- Далі відбувається завантаження сцени LoadingScreen за допомогою функції Application.LoadLevel("LoadingScreen").

2.2 Структура підпрограми: гараж

Постановка завдання: визначити, які цілі має виконувати гараж у грі X-Racer.

Реалізація поставленного завдання: згідно з визначеними цілями, а саме створення відділу детейлінгу та стайлінгу автомобіля, перегляд наявних та доступних для придбання автомобілів.

Механіка гаража дає гравцеві свободу вибору та налаштування автомобілів, що дозволяє створити унікальний автопарк, відобразити свій стиль та покращити продуктивність машин для досягнення успіху у грі.

Гараж у грі є спеціальним розділом, де гравець може керувати своїми автомобілями, робити покупки нових автомобілів і налаштовувати їх зовнішній вигляд і характеристики.

Після відкриття гаража гравець бачить:

1. Придбані автомобілі:

- "Придбані" гравець може побачити список усіх автомобілів, які він уже придбав у грі.
- Кожен автомобіль представлений з назвою та зображенням, дозволяючи гравцеві легко визначити, який автомобіль хоче вибрати.

2. Доступні для придбання автомобілі:

- Гравець може переглянути доступні для покупки автомобілі.
- Кожен автомобіль супроводжується інформацією про його характеристики, вартість та зображення, що допомагає гравцеві прийняти рішення про покупку.

2.2.1 Додатковий функціонал:

Постановка завдання: створення другорядного функціоналу, який надасть можливість фотографування автомобілів у різних куточках гаражу.

Фоторежим

- У гаражі також є фоторежим, який дозволяє гравцеві зробити красиві знімки своїх автомобілів.
- Після натискання кнопки фоторежиму (фотоапарат) машина відображається на подіумі з можливістю повороту її модельки.

Реалізована функція виглядає таким чином:

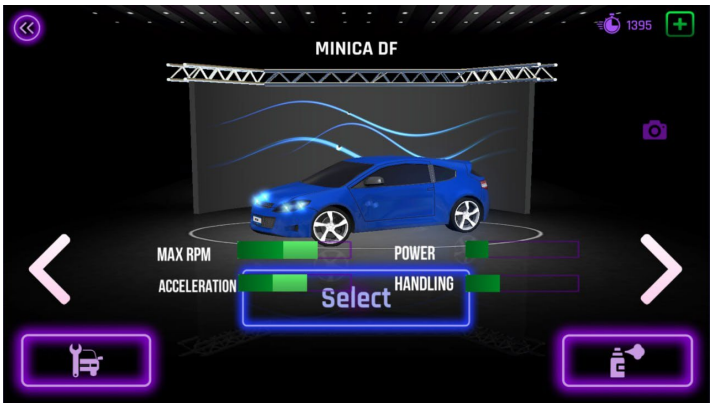


Рисунок 2.7 Вигляд до натискання

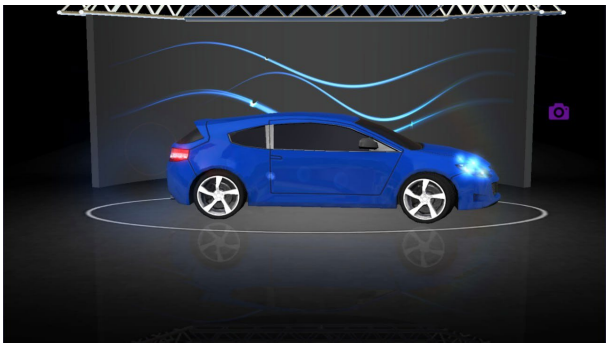


Рисунок 2.8 Вигляд після натискання

2.2.2 Стайлінг

- У гаражі гравець має можливість налаштовувати зовнішній вигляд своїх автомобілів, включаючи зміну кольору кузова.
- За допомогою вибору конкретного відтінку гравець може змінити колір автомобіля на той, який відповідає його перевагам та стилю;
- Спочатку гравцеві доступний 1 безкоштовний колір, решта вимагають оплати внутрішньоігровою валютою.

Рисунок 2.9 До стайлінгу

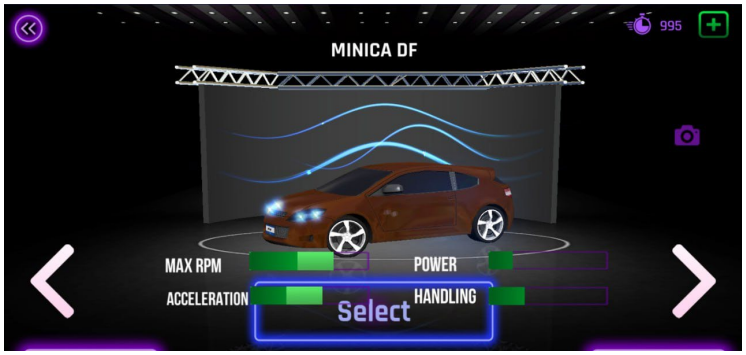


Рисунок 2.10 Після стайлінгу

Нижче представлена програмна побудова механіки зміни кольору (рис. 2.11):

```
public class HR_ModificationColor : MonoBehaviour {

    public pickedColor _pickedColor;
    public enum pickedColor{Orange, Red, Green, Blue, Yellow, Black, White, Cyan, Magenta, Pink}
    public int colorPrice;
    public bool unlocked = false;
    private Text priceLabel;
    private Image priceImage;
```

Рисунок 2.11 Керування кольорами для модифікації

Код є скриптом з назвою HR_ModificationColor, який, відповідає за керування кольорами для модифікації об'єктів у грі.

Ось його основні характеристики:

Перелік pickedColor: Визначаються доступні кольори, які можна вибрати для модифікації об'єктів. Значення включають "Orange", "Red", "Green", "Blue", "Yellow", "Black", "White", "Cyan", "Magenta" та "Pink".

Змінна _pickedColor: Зберігає поточний вибраний колір із переліку pickedColor.

Змінна colorPrice: Ціна кольору. Використовується для встановлення вартості вибраного кольору.

Змінна unlocked: Визначає, чи колір розблокований. Використовується для контролю доступності вибору певного кольору.

Приватні змінні priceLabel та priceImage: Відповідають за відображення ціни кольору.

Продовження програмної структури:

```
public void OnClick () {

    if(!unlocked){
        BuyColor();
        return;
    }

    HR_ModHandler handler = GameObject.FindObjectOfType<HR_ModHandler>();
    Color selectedColor = new Color();

    switch(_pickedColor){

    case pickedColor.Orange:
        selectedColor = Color.red + (Color.green / 2f);
        break;

    case pickedColor.Red:
        selectedColor = Color.red;
        break;

    case pickedColor.Green:
        selectedColor = Color.green;
        break;

    case pickedColor.Blue:
        selectedColor = Color.blue;
        break;

    case pickedColor.Yellow:
        selectedColor = Color.yellow;
        break;

    case pickedColor.Black:
        selectedColor = Color.black;
        break;
```

Рисунок 2.12 Обробник події натискання на кнопку

```
    case pickedColor.White:
        selectedColor = Color.white;
        break;

    case pickedColor.Cyan:
        selectedColor = Color.cyan;
        break;

    case pickedColor.Magenta:
        selectedColor = Color.magenta;
        break;

    case pickedColor.Pink:
        selectedColor = new Color(1, 0f, .5f);
        break;

    }

    handler.ChangeChassisColor(selectedColor);

}
```

Рисунок 2.13 Обробник події натискання на кнопку

Цей код є методом OnClick() є обробником події натискання на кнопку.

Ось його основна дія:

1) Спочатку перевіряється значення змінної unlocked. Якщо воно дорівнює false, то викликається метод BuyColor(), яка відповідає за покупку кольору. Потім виконання методу переривається за допомогою оператора return, щоб запобігти виконанню решти коду.
2) Якщо значення unlocked дорівнює true, то створюється екземпляр класу HR_ModHandler, використовуючи метод GameObject.FindObjectOfType<HR_ModHandler>(). Це означає, що HR_ModHandler відповідає за керування зміною кольору.
3) Створюється змінна selectedColor типу Color, і їй присвоюється початкове значення порожнього кольору.
4) За допомогою оператора switch перевіряється значення _pickedColor (перелік pickedColor). Залежно від значення _pickedColor, змінної selectedColor надається відповідне значення кольору. Наприклад, для кожного випадку в блокі switch визначається конкретний колір, який буде присвоєний змінній selectedColor.
5) Метод ChangeChassisColor(selectedColor) викликається у екземпляра класу HR_ModHandler для зміни кольору шасі або іншого елемента. Цей метод реалізує зміну кольору об'єкта, ґрунтуючись на вибраному кольорі selectedColor.

Таким чином, при натисканні на кнопку або інший елемент інтерфейсу метод OnClick() перевірятиме значення unlocked, а потім, залежно від вибраного кольору, змінювати колір відповідного елемента за допомогою методу ChangeChassisColor() у екземпляра класу HR_ModHandler.

```
void Update(){

    string currentColorString = _pickedColor.ToString();

    if(colorPrice <= 0 && !unlocked){
        PlayerPrefs.SetInt(GameObject.FindObjectOfType<RCC_CarControllerV3>().transform.name + "OwnedColor" + currentColorString, 1);
        unlocked = true;
    }

    if(PlayerPrefs.HasKey(GameObject.FindObjectOfType<RCC_CarControllerV3>().transform.name + "OwnedColor" + currentColorString))
        unlocked = true;
    else
        unlocked = false;

    if(!unlocked){
        if(!priceImage.gameObject.activeSelf)
            priceImage.gameObject.SetActive(true);
        if(priceLabel.text != colorPrice.ToString())
            priceLabel.text = colorPrice.ToString();
    }else{
        if(priceImage.gameObject.activeSelf)
            priceImage.gameObject.SetActive(false);
        if(priceLabel.text != "UNLOCKED")
            priceLabel.text = "UNLOCKED";
    }

}
```

Рисунок 2.14 Викликається кожен кадр оновлення стану об`єкта

Цей код представляє метод Update(), який, викликається кожен кадр оновлення стану об'єкта.

Ось його основна дія:

1) Спочатку створюється рядкова змінна currentColorString, якій надається рядкове уявлення значення _pickedColor. Це дозволяє отримати ім'я поточного вибраного кольору.
2) Потім відбувається перевірка умови colorPrice <= 0 && !unlocked. Якщо ціна кольору (colorPrice) менша або дорівнює нулю і прапор unlocked має значення false, то виконується таке: використовуючи клас PlayerPrefs, встановлюється значення ключа, що складається з імені об'єкта RCC_CarControllerV3 та імені поточного кольору (currentColorString), що дорівнює 1. Це означає, що колір був розблокований. Значення прапора unlocked встановлюється в true, щоб відобразити розблокування кольору.

3) Потім відбувається перевірка наявності ключа PlayerPrefs. Якщо існує ключ, що складається з імені об'єкта RCC_CarControllerV3 і імені поточного кольору (currentColorString), прапор unlocked встановлюється в true, в іншому випадку - в false.
4) Далі виконується перевірка значення прапора unlocked. Якщо unlocked має значення false, то відбувається таке: якщо компонент priceImage неактивний (activeSelf == false), він стає активним (SetActive(true)). Це відображає зображення ціни. Якщо текст priceLabel не дорівнює рядковому поданню colorPrice, йому присвоюється нове значення colorPrice.ToString(). Це відображає ціну кольору.
5) Якщо прапор unlocked має значення true, то виконується таке:

Якщо компонент priceImage активний (activeSelf == true), він стає неактивним (SetActive(false)). Ймовірно, це приховує зображення ціни.

Якщо текст priceLabel не дорівнює "UNLOCKED", йому присвоюється нове значення "UNLOCKED". Це відображає, що колір розблоковано. Таким чином, метод Update() оновлює стан об'єкта відповідно до розблокування кольору та його ціни, відображаючи відповідні елементи інтерфейсу залежно від цього стану.

```
    void BuyColor(){

        int playerCoins = PlayerPrefs.GetInt("Currency");
        string currentColorString = _pickedColor.ToString();

        if(playerCoins >= colorPrice){
            HR_ModHandler handler = GameObject.FindObjectOfType<HR_ModHandler>();
            handler.BuyProperty(colorPrice, GameObject.FindObjectOfType<RCC_CarControllerV3>().transform.name + "OwnedColor" + currentColorString);
        }

    }
}
```

Рисунок 2.15 Функція, яка виконується при спробі купівлі кольору

Цей код є методом BuyColor(), який виконується при спробі купівлі кольору. Ось його основна дія:

1) Спочатку створюється ціла змінна playerCoins, якій присвоюється значення, отримане з PlayerPrefs під ключом "Currency". Це становить кількість монет у гравця.
2) Потім створюється рядкова змінна currentColorString, якій надається рядкове уявлення значення _pickedColor. Це дозволяє отримати ім'я поточного вибраного кольору.
3) Відбувається перевірка умов playerCoins >= colorPrice. Якщо кількість монет гравця більша або дорівнює ціні кольору (colorPrice), виконується таке:

Створюється екземпляр класу HR_ModHandler, знайденого методом FindObjectOfType. Це обробник модифікацій.

Викликається метод BuyProperty() у обробника модифікацій, передаючи йому ціну кольору (colorPrice) та ключ, що складається з імені об'єкта RCC_CarControllerV3 та імені поточного кольору (currentColorString). Він купує кольори і зберігає інформацію про володіння кольором.

4) Якщо умова playerCoins >= colorPrice не виконується, ніякі дії не виконуються, і купівля кольору не відбувається.

Таким чином, метод BuyColor() перевіряє, чи достатньо гравця монет для покупки кольору, і якщо так, виконує покупку через екземпляр HR_ModHandler.

2.2.3 Детейлінг

- На екрані дітей лінгу гравець може налаштовувати деталі автомобіля для покращення його продуктивності.
- Доступні деталі включають турбіни, налаштування двигуна, поршні, коробку передач, підвіску та колеса.
- Кожна деталь пропонує кілька варіантів або рівнів покращень, дозволяючи гравцеві створювати унікальні та потужні автомобілі відповідно до своїх уподобань та стилю гри. Базовий комплект безкоштовний, для подальших покращень потрібно внутрішньоігрова валюта.

Рисунок 2.16 Покращена турбіна

Програмна реалізація функції детейлінгу:

```
void BuyUpgrade(){

    int playerCoins = PlayerPrefs.GetInt("Currency");
    HR_ModApplier applier = GameObject.FindObjectOfType<HR_ModApplier>();
    HR_ModHandler handler = GameObject.FindObjectOfType<HR_ModHandler>();

    if(playerCoins >= upgradePrice){

        switch(upgradeClass){

        case UpgradeClass.Speed:
            if(applier.speedLevel < 5){
                handler.UpgradeSpeed();
                PlayerPrefs.SetInt("Currency", playerCoins - upgradePrice);
            }
            break;
        case UpgradeClass.Handling:
            if(applier.handlingLevel < 5){
                handler.UpgradeHandling();
                PlayerPrefs.SetInt("Currency", playerCoins - upgradePrice);
            }
            break;
        case UpgradeClass.Brake:
            if(applier.brakeLevel < 5){
                handler.UpgradeBrake();
                PlayerPrefs.SetInt("Currency", playerCoins - upgradePrice);
            }
            break;
        case UpgradeClass.Siren:
            if(applier.isSirenPurchased == false){
                handler.UpgradeSiren();
                PlayerPrefs.SetInt("Currency", playerCoins - upgradePrice);
            }
            break;

        }
```

Рисунок 2.17 - Функція BuyUpgrade(), яка відповідає за купівлю покращень

Оголошується змінна playerCoins і їй надається значення поточної валюти гравця, отриманої з PlayerPrefs за допомогою PlayerPrefs.GetInt("Currency").

Знаходиться екземпляр класу HR_ModApplier і надається змінною applier за допомогою GameObject.FindObjectOfType<HR_ModApplier>(). Це дозволяє отримати доступ до методів та властивостей цього класу.

Знаходиться екземпляр класу HR_ModHandler і надається змінною handler за допомогою GameObject.FindObjectOfType<HR_ModHandler>(). Це дозволяє отримати доступ до методів та властивостей цього класу.

Використовується оператор switch для перевірки значення змінної upgradeClass.

Якщо значення upgradeClass дорівнює UpgradeClass.Speed, код виконується всередині цього блоку case. Далі перевіряється, що рівень швидкості (applier.speedLevel) менше 5. Потім викликається метод UpgradeSpeed() об'єкта handler і значення валюти гравця (playerCoins) зменшується на ціну апгрейду (upgradePrice). Потім нове значення валюти зберігається в PlayerPrefs за допомогою PlayerPrefs.SetInt("Currency", playerCoins - upgradePrice).

2.3 Модуль механіки: управління гравцем кнопками

Постановка завдання: визначити, які дії гравця будуть доступні; за допомогою яких кнопок гравець керуватиме персонажем та призначення функцій кнопкам.

При розробці механіки такого типу управління важливо брати до уваги зручність та інтуїтивність. Добре збалансована та ефективна механіка управління може суттєво покращити геймплей та загальний досвід користувача.

Реалізоване завдання виглядає таким чином:

Пропонує гравцям можливість контролювати автомобіль за допомогою віртуальних кнопок, що включають газ, гальмо та нітро:

Кнопка газу: Гравець може натиснути кнопку газу, щоб збільшити швидкість автомобіля. При утриманні кнопки газу автомобіль продовжуватиме рух із встановленою швидкістю, доки гравець не відпустить кнопку.

Рисунок 2.18 Зображення кнопки гальма

Кнопка гальма: Натискання кнопки гальма дозволяє гравцеві зменшити швидкість або зупинити автомобіль повністю. При утриманні кнопки гальма автомобіль сповільнюватиметься, а при відпусканні кнопки швидкість відновлюватиметься. Кнопка гальма дозволяє гравцеві контролювати швидкість та робити точні маневри на трасі.

```
void ApplyBrakeTorque(RCC_WheelCollider wc, float brake){

    if(ABS && handbrakeInput <= .1f){

        WheelHit hit;
        wc.wheelCollider.GetGroundHit(out hit);

        if((Mathf.Abs(hit.forwardSlip) * Mathf.Clamp01(brake)) >= ABSThreshold){
            ABSAct = true;
            brake = 0;
        }else{
            ABSAct = false;
        }

    }

    if(speed < minimumSpeed)
        brake = 0;

    wc.wheelCollider.brakeTorque = brake;

}
```

Рисунок 2.19 Метод використовується для застосування гальмівної крутного моменту до колеса автомобіля.

```
void Braking (){

    //Handbrake
    if(handbrakeInput > .1f){
        ApplyBrakeTorque(RearLeftWheelCollider, (brake * 1.5f) * handbrakeInput);
        ApplyBrakeTorque(RearRightWheelCollider, (brake * 1.5f) * handbrakeInput);

    }else{
        // Braking.
        ApplyBrakeTorque(FrontLeftWheelCollider, brake * (Mathf.Clamp(_brakeInput, 0, 1)));
        ApplyBrakeTorque(FrontRightWheelCollider, brake * (Mathf.Clamp(_brakeInput, 0, 1)));
        ApplyBrakeTorque(RearLeftWheelCollider, brake * Mathf.Clamp(_brakeInput, 0, 1) / 2f);
        ApplyBrakeTorque(RearRightWheelCollider, brake * Mathf.Clamp(_brakeInput, 0, 1) / 2f);

    }

}
```

Рисунок 2.20 Метод використовується для виконання гальмування автомобіля

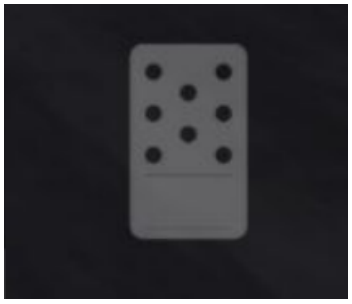


Рисунок 2.21 Кнопка газу

Гра надає кнопки управління вліво та вправо. Гравець може натиснути кнопку вліво для повороту автомобіля вліво та кнопку вправо для повороту вправо. Ці кнопки дозволяють гравцеві здійснювати точні маневри на трасі та обганяти суперників.

Рисунок 2.22 Стрілочки для управління сенсором

Постановка завдання: визначити, яким чином плануємо реалізувати "нітро" у грі і як воно буде впливати на геймплей.

Реалізація додаткового функціоналу:

Кнопка нітро: Натискання кнопки нітро активує спеціальний режим прискорення, який дає автомобілю додатковий приріст швидкості на обмежений час. При використанні нітро гравець може випередити суперників або швидко подолати ділянки траси з перешкодами. Після використання нітро автомобіль відновлюється у звичайний режим руху.

У грі передбачений спеціальний індикатор нітро, розташований на верхній панелі під спідометром.

Цей індикатор дозволяє гравцеві відстежувати рівень нітро і знати, скільки часу залишилося до його повного використання або відновлення.

Індикатор подається у вигляді графічного елемента (шкали), які заповнюються або спустошуються залежно від доступної кількості нітро. При активації нітро-кнопка отримує анімацію хвилі, а користувач відчуває спеціальні ефекти (звук та анімацію полум'я з вихлопної труби).

Такий індикатор нітро допомагає гравцям стратегічно планувати використання нітро та приймати рішення про те, коли краще активувати його для досягнення найкращого результату на трасі.

Візуальна реалізація:

Рисунок 2.23 Позначка прискорювача

2.4 Модуль механіки: управління гравцем нахилом

Постановка завдання: функціонал механіки повинен включати реєстрацію та відстеження віртуальних осей та кнопок, а також отримання їх стану.

Акселерометр - це прилад, який вимірює проекцію прискорення (різниці між істинним прискоренням об'єкта і гравітаційним прискоренням). Акселерометр дозволяє фіксувати навіть незначні зміни прискорення залежно від його положення у просторі.

Реалізація поставленного завдання:

Головна ідея полягає в тому, що нахил пристрою використовується як метод управління рухами гравця або об'єкта у грі.

Основною перевагою цієї механіки є її інтуїтивність. Замість використання традиційних контролерів, гравець може просто нахилити пристрій у потрібному напрямку, що надає більш природний та гнучкий спосіб керування. Це особливо корисно у випадку гри з 3D-середовищем, де нахил може допомогти гравцеві контролювати камеру або рухатися вдоль тривимірного простору.

Загалом, модуль механіки управління гравцем нахилом відкриває нові можливості для інтерактивності та взаємодії з ігровим світом, надаючи користувачам більш імерсивний ігровий досвід.

2.5 Модуль механіки: ігрова валюта

Постановка завдання:

У грі реалізовано механіку монет (грошей), яка додає елемент прогресії та різноманітності в ігровий процес.

```
int dailyRewardValue(int index) //get daily reward value
{
    int[] rewardValue = new int[] { 50, 100, 200, 500, 800, 1000 };
    return rewardValue[index];
}
public void DailyBonus(int day) //daily bonus btn clicked
{
    int reward = dailyRewardValue(day - 1);
    PlayerPrefs.SetInt("Currency", PlayerPrefs.GetInt("Currency") + reward);
    PlayerPrefs.SetInt("BONUSDAY", day + 1);
    if (userName != null)
    {
        dailyRewardUserName.text = "Hey! " + userName;
    }
    dailyRewardDays[day - 1].interactable = false;
    dailyRewardSubPanel.SetActive(true);
}
```

Рисунок 2.24 Функція щоденної нагороди

Гравці можуть заробляти гроші, виконуючи різні завдання, виконання маневрів або перемагаючи у гонках. Чим складніше завдання або гонка, тим більше винагорода буде отримана.

Вони представлені у вигляді віртуальної валюти, яку гравці можуть використовувати для різних цілей, таких як покупка нових автомобілів, покращень, кастомізації та розблокування нових рівнів та трас.

Рисунок 2.25 Віртуальна валюта

У грі передбачений магазин, де гравці можуть купувати різноманітні предмети, включаючи нові автомобілі різних класів та покращення для автомобілів. Кожен предмет має свою ціну, і гравці повинні мати достатньо монет, щоб їх придбати.

```
public void BuyCar (){

        if(PlayerPrefs.GetInt("Currency") >=
HR_PlayerCars.Instance.cars[carIndex].price)
            PlayerPrefs.SetInt("Currency", PlayerPrefs.GetInt("Currency") -
HR_PlayerCars.Instance.cars[carIndex].price);
        else
            return;

        PlayerPrefs.SetInt(cars[carIndex].name + "Owned", 1);

        SpawnCar();

    }
```

Рисунок 2.26 Метод BuyCar(), який викликається для покупки автомобіля у грі

2.5.1 Прогресія та розблокування:

Деякі автомобілі, траси або ігрові режими можуть бути заблоковані на початку гри. Гравці повинні заробляти та накопичувати гроші, щоб розблокувати нові можливості та контент у грі. Це створює почуття прогресії та заохочує гравців продовжувати грати та розвиватися.

Рисунок 2.27 Недоступні рівні

2.5.2 Бонусні поїнти

Постановка завдання:

1) Створити логіку, яка буде винагороджувати гравця цінними очками за виконання різних дій та досягнень під час гонки, таких як унікальні маневри, трюки, завдання та тривалість тримання високої швидкості;

2) Реалізувати систему обліку рахунку гравця: розрахувати оцінку продуктивності гравця на основі різних факторів, таких як година проходження, навички управління, виконані трюки та завдання. Забезпечити відображення цього рахунку у таблиці зароблених грошей.

3) Реалізувати систему обліку відстані: відстежувати загальну пройдену дистанцію під час гонки. Забезпечити відображення цього параметра в таблиці зароблених грошей.

4) Впровадити систему майже зіткнень: визначити ситуації, коли гравець уникає зіткнення з іншими автомобілями або перешкодами на трасі, та винагородити його додатковими бонусними очками. Забезпечити відображення цього параметра в таблиці зароблених грошей.

5) Розробити систему винагород за високу швидкість: визначити моменти, коли автомобіль рухається зі швидкістю вище певного порога (наприклад, 100 км/год) та винагородити гравця бонусними очками або додатковими винагородами. Забезпечити відображення цього параметра в таблиці зароблених грошей.

6) Створити таблицю зароблених грошей: після закінчення кожної гонки відобразити гравцю таблицю з детальними результатами, включаючи загальний рахунок, пройдене відстань, кількість майже зіткнень та

швидкість автомобіля. Забезпечити чітке візуальне подання цих даних для зручного аналізу гравцем.

Реалізація поставленного завдання:

Кожна дія та досягнення, виконана під час гонки, буде винагороджена цінними очками, що дозволяють гравцеві відстежувати свій прогрес. Ось деякі особливості процесу нарахування очок:

Маневри: Щоразу, коли гравець робить унікальний або небезпечний маневр, чи то складний поворот або прохід поряд з перешкодою, дії будуть винагороджені додатковими очками.

Чим довше користувач тримає високу швидкість, ви проходите гонку і досягаєте високих швидкостей, тим більше очок нараховується.

Також у грі передбачені завдання, які дозволяють заробити додаткові очки.

Після закінчення кожної гонки гравцю подається таблиця із заробленими грошима. Ця таблиця служить для огляду та аналізу фінансових результатів перегонів. Ось основні особливості таблиці із заробленими нагородами:

Your Score (Ваш рахунок): Це показник, який відображає загальну оцінку продуктивності гравця під час перегонів. Він розраховується з урахуванням різних чинників, включаючи час проходження, навички управління, кількість виконаних трюків, завдань та інших параметрів, що з ігровим процесом. Чим вище рахунок, тим краща продуктивність гравця.

Distance (Відстань): Це значення вказує на загальну пройдену дистанцію гравцем під час перегонів. Воно вимірюється в ігрових одиницях, таких як метри або кілометри, і відображає, наскільки далеко гравець просунувся трасою.

Near miss (Майже зіткнення): Цей термін стосується ситуацій, коли гравець здійснив майже зіткнення з іншим автомобілем, перешкодою або об'єктом на трасі, але зміг уникнути зіткнення в останній момент. Чим більше таких зіткнень вдалося уникнути, тим більше бонусних очок і можливо додаткових грошових нагород отримає гравець.

Above 100 / H (Вище 100 / год): Цей вираз вказує на швидкість автомобіля в моменти гонки, коли він рухається зі швидкістю вище 100 одиниць на годину (або іншої одиниці вимірювання швидкості, прийнятої у грі). Це може бути важливим фактором для отримання бонусних очок або додаткових винагород, пов'язаних із високою швидкістю та динамічним стилем гри.

Таким чином, таблиця із зароблениими грошима після гонки надає гравцям повну інформацію про їхній результат, включаючи їхній загальний рахунок, пройдену відстань, кількість майже зіткнень та досягнень, а також інформацію про швидкість автомобіля. Це допомагає гравцям аналізувати свої досягнення та прогрес у грі.

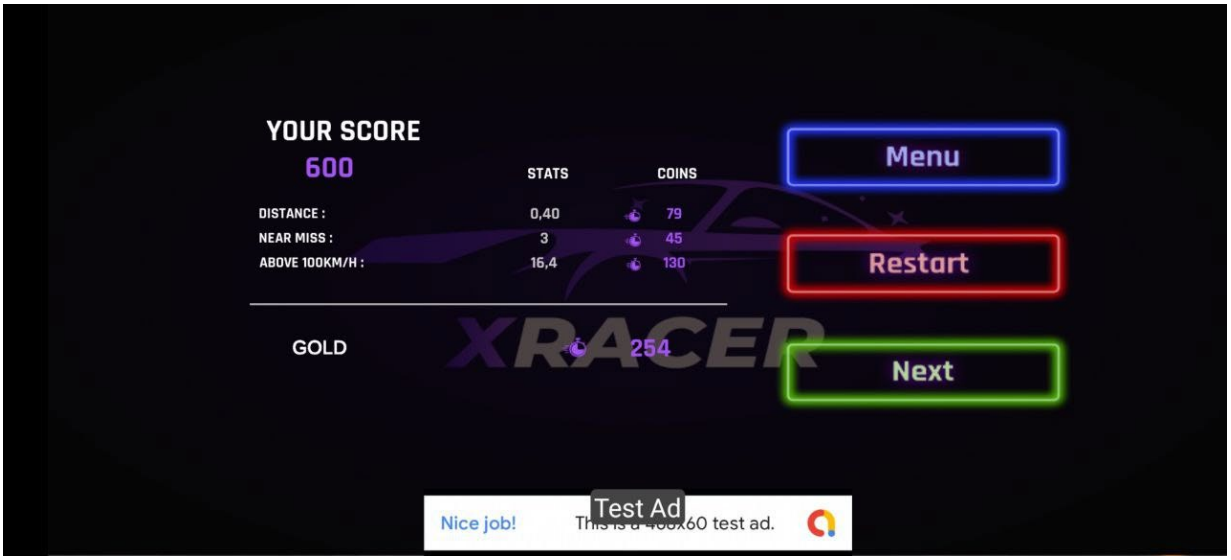


Рисунок 2.28 Таблиця досягнень після завершення проходження

2.6 Модуль механіки: трафік автомобілів для перешкод

Постановка завдання:

- визначити кількість типів ботів з унікальними характеристиками та поведінкою, такими як агресивність, стратегія гри, рівень тяжкості.

Реалізація поставленного завдання:

У грі присутні боти, які є комп'ютерно керованими автомобілями, що створюють додаткову динаміку та виклик для ігрового досвіду. Основні характеристики та функції ботів у грі:

Інтелект ботів: Боти мають програмний інтелект, який дозволяє їм приймати рішення та адаптуватися до умов дорожнього руху. Вони можуть реагувати на дії гравця, інших ботів та довкілля. Інтелект ботів може змінюватись в залежності від рівня складності гри, створюючи різні рівні виклику для гравця.

Різноманітність ботів: У грі є різні типи ботів, включаючи вантажні автомобілі, легкові автомобілі, автобуси, поліцію та машини для завдань. Кожен тип ботів має свої характеристики та поведінку на дорозі. Вантажні автомобілі можуть рухатися повільніше, але мати більшу потужність, тоді як легкові автомобілі можуть бути більш маневреними та швидкими. Це створює різноманітність ситуацій та викликів для гравця.

Реалістична поведінка ботів: Боти у грі мають реалістичну поведінку, що відповідає типу автомобіля. Боти також можуть виконувати маневри, такі як обгін, зміна смуги руху та гальмування, створюючи динамічну та реалістичну обстановку на дорозі.

Інтеракція з ботами: Гравець може взаємодіяти з ботами у грі, включаючи обгін, поступання дороги або прямування за ними. Це може знадобитися для виконання певних завдань або просто для успішного проходження рівнів. Взаємодія з ботами може створювати додаткові виклики та вимагати від гравця стратегічного мислення та навичок управління.

Загалом наявність ботів у грі надає їй більшої динаміки, реалістичності та створює різноманітність ситуацій, з якими гравець може зіткнутися на дорозі. Вони представляють додатковий виклик та роблять ігровий процес більш захоплюючим та цікавим.

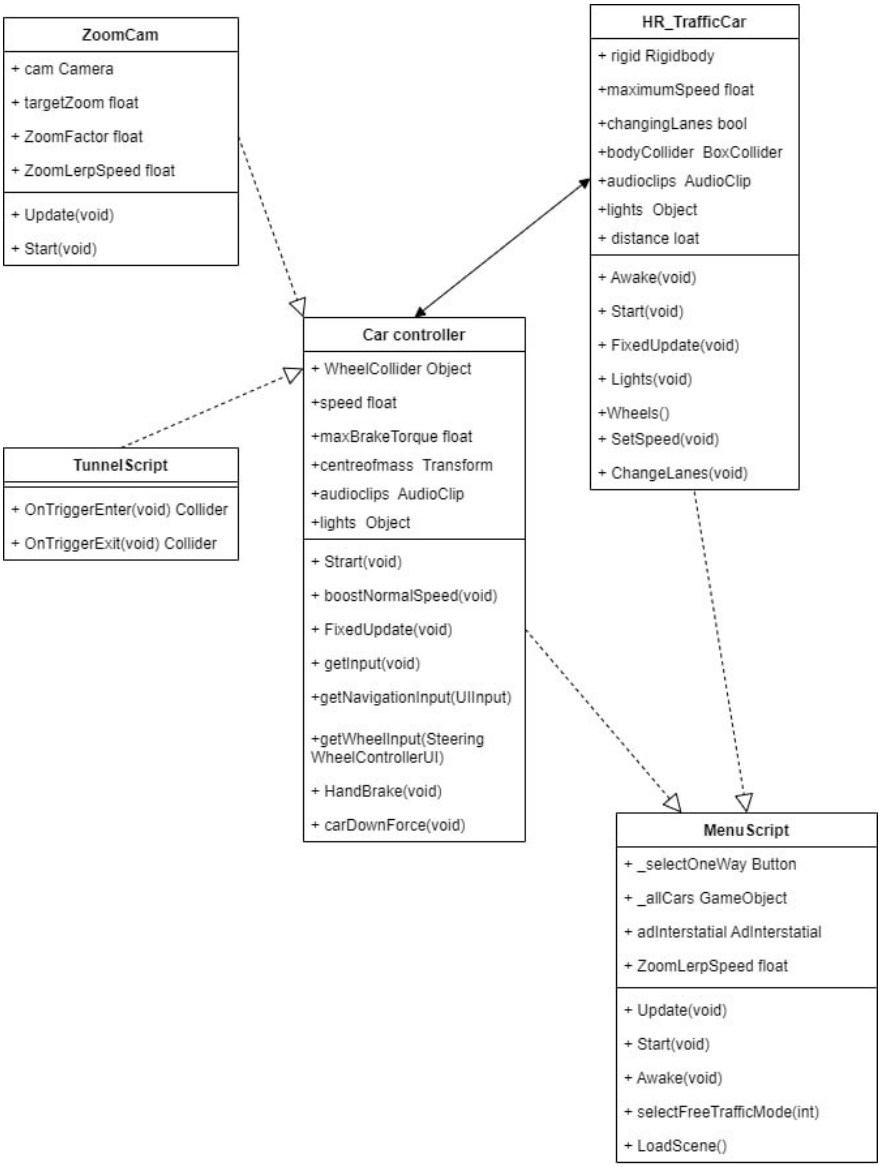


Рисунок 2.29 – Схема зв`язків між класами

Програмна реалізація механіки (рис. 2.29):

```
void CreateTraffic()
{


    for (int i = 0; i < density; i++)
    { //trafficCars.Length(replaced)

        for (int k = 0; k < trafficCars[i].frequence; k++)
        {
            GameObject go = (GameObject)GameObject.Instantiate(trafficCars[i].trafficCar, trafficCars[i].trafficCar.transform.position,
            trafficCars[i].trafficCar.transform.rotation);
            _trafficCars.Add(go.GetComponent<HR_TrafficCar>());
            go.SetActive(false);
            // only level two // List Index 7,14
            if (LevelChallange._level == 11)
            {
                if (go.transform.GetChild(1).gameObject.tag == "Destroycar")
                {
                    go.transform.GetChild(1).gameObject.SetActive(true);
                }
            }
        }

    }
}
```

Рисунок 2.29 - Функція CreateTraffic(), яка відповідає за створення транспортного руху

1. Починається функція CreateTraffic(), яка відповідає за створення транспортного руху.
2. Зовнішній цикл for проходить через кількість разів, визначену змінною density. Це вказує на те, скільки типів транспорту буде створено.
3. У середині зовнішнього циклу знаходиться ще один цикл for, який виконується trafficCars[i].frequence разів. Це вказує на кількість автомобілів, які будуть створені для кожного типу транспорту.
4. У середині іншого циклу створюється новий об'єкт гри (GameObject) на основі префабу trafficCars[i].trafficCar. Позиція та обертання нового об'єкта беруться з відповідних значень префабу.
5. Отриманий об'єкт гри додається до списку _trafficCars за допомогою функції go.GetComponent<HR_TrafficCar>(). Це передбачає, що

компонент HR_TrafficCar прикріплений до префабу транспортного засобу.

6. Об'єкт гри виключається (SetActive(false)), що означає, що він неактивний та невидимий у грі.
7. Умова if (LevelChallange._level == 11) перевіряє, чи поточний рівень гри дорівнює 11. Якщо так, то виконується додатковий код. У цьому випадку, для об'єкта гри, який має дочірній об'єкт із тегом "Destroycar" (знищення автомобіля), дочірній об'єкт стає активним (SetActive(true)).

Далі розглянемо функцію “AnimateTraffic”, яка відповідає за анімацію руху транспортних засобів у грі.

```
void AnimateTraffic()
{
    for (int i = 0; i < _trafficCars.Count; i++)
    {
        if (_trafficCars[i].isAnimating == false)
        {
            if (_player.transform.position.z > (_trafficCars[i].transform.position.z + 15) ||
            _player.transform.position.z < (_trafficCars[i].transform.position.z - (325)))
            {
                ReAlignTraffic(_trafficCars[i]);
            }
        }

    }
}
```

Рисунок 2.30 Метод AnimateTraffic

1. Цикл for проходить по кожному транспортному засобу у списку _trafficCars.
2. Умова перевіряє, чи єанімація транспортного засобу рівна false, що означає, що він не знаходиться в стані анімації.

3. Умова перевіряє, чи позиція гравця по вісі Z (_player.transform.position.z) знаходиться за межами певного діапазону від позиції транспортного засобу (_trafficCars[i].transform.position.z). Конкретні значення цього діапазону вказуються як +15 і -325 відповідно.
4. Якщо умова виконується, викликається функція ReAlignTraffic із параметром _trafficCars[i]. Це призводить до переналаштування позиції та стану транспортного засобу.

Далі розглянемо функцію “CheckIfClipping”, яка перевіряє, чи транспортний засіб зазнає зіткнення з іншими транспортними засобами у грі.

```
bool CheckIfClipping(BoxCollider trafficCarBound)
{

    for (int i = 0; i < _trafficCars.Count; i++)
    {
        if (_trafficCars[i].isAnimating == false)
        {
            if (!trafficCarBound.transform.IsChildOf(_trafficCars[i].transform) && _trafficCars[i].gameObject.activeSelf)
            {
                if (HR_BoundsExtension.ContainBounds(trafficCarBound.transform,
                trafficCarBound.bounds, _trafficCars[i].triggerCollider.bounds))
                {
                    return true;
                }
            }
        }
    }

    return false;

}
```

Рисунок 2.31 CheckIfClipping

1. Функція приймає параметр trafficCarBound, який є BoxCollider транспортного засобу, який перевіряється на зіткнення.
2. Цикл for проходить по кожному транспортному засобу у списку _trafficCars.

3. Умова перевіряє, чи єанімація транспортного засобу рівна false, що означає, що він не знаходиться в стані анімації.
4. Умова перевіряє, чи trafficCarBound не є дочірнім об'єктом транспортного засобу (!trafficCarBound.transform.IsChildOf(_trafficCars[i].transform)) і чи транспортний засіб активний (_trafficCars[i].gameObject.activeSelf).
5. Умова перевіряє, чи колайдер trafficCarBound.bounds транспортного засобу перетинається з колайдером triggerCollider.bounds транспортного засобу _trafficCars[i] за допомогою функції HR_BoundsExtension.ContainBounds.
6. Якщо умова виконується для будь-якого транспортного засобу, то функція повертає true, що означає, що зіткнення відбувається.
7. Якщо цикл пройшов усі транспортні засоби і жодного зіткнення не виявлено, функція повертає false, що означає відсутність зіткнень.

Функція CheckIfClipping використовує колайдери для виявлення столкнень між транспортними засобами у грі.

3 ТЕСТУВАННЯ

3.1 Тестування методом «чорної скриньки». Тест кейси

Суть полягає у виконанні деякої кількості дій та/або умов, необхідних для перевірки певної функціональності програмної системи, що розробляється.

Проблемні тестові випадки:

- Які залежні один від одного (наприклад, очікується, що частина кроків виконана в попередньому test case, є посилання на інші test cases)
- З нечітким формулюванням кроків
- З нечітким формулюванням ідеї або очікуваного результату

Test cases можуть бути:

- Позитивні – використовуються тільки коректні дані і перевіряють, чи правильно додаток виконує функцію
- Негативні – використовуються як коректні, так і некоректні дані і ставить за мету перевірку виняткових ситуацій, а також перевіряє, що функція не виконується при спрацьовування валідатора

Основні стани проходження тест кейсу:

- No run (не запущено)
- Passed (пройдено)
- Failed (помилка)
- Blocked (заблоковано)
- Not Completed (не завершено)

Таблиця 1 – Тест кейси

Опис тестового випадку	**Кроки для відтворення**	**Очікуваний результат**	**Статус**
Відкриття головного меню гри	1.Натиснути на значок гри на робочому столі 2. Звернiути увагу на екран.	Перед гравцем відкривається головний екран гри після запуску програми.	Пройдений
Вихід із гри.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку вимкнення; 3. Натиснути на кнопку "Yes".	Гравець успішно виходить із гри після натискання на кнопку "Yes".	Пройдений
Повернення до головного меню.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку вимкнення; 3. Натиснути на кнопку "No".	Гравець успішно повертається до головного меню після натискання на кнопку "No".	Пройдений
Перехід до налаштувань .	1. Запустити гру "XRacer"; 2. Натиснути на кнопку "гайковий ключ" .	Гравцю відкривається меню налаштувань гри після натискання на кнопку налаштувань (гайковий ключ).	Пройдений
Увімкнення/вимкнення звуку.	1. Запустити гру "XRacer"; 2 Натиснути на кнопку налаштувань	Гравець регулює увімкнення/вимкнення звуку в грі за допомогою натискання на	Пройдений

	"гайковий ключ"; 3. Натиснути на тумблер "Sound"; 4. Звернути увагу, що звук вимкнувся; 5. Повторно Натиснути на тумблер "Sound"; 6. Звернути увагу, що звук увімкнувся.	тумблер "Sound" в налаштуваннях.	
Регулювання графіка гри.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку налаштувань "гайковий ключ"; 3. Вибрати 1 із 3 варіантів графіки: Low/Medium/High.	Гравець регулює якість графіки у грі за допомогою натискання на кнопки "Graphics Quality" у налаштуваннях.	Пройдений
Зміна управління у грі.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку налаштувань "гайковий ключ"; 3. Вибрати 1 з 2 типів управління: Touch/Tilt (Auto).	Гравець змінює керування автомобілем з кнопкового на автоматичний (за допомогою повороту пристрою) завдяки натисканням на кнопки "Control" в налаштуваннях.	Пройдений

Повернення до головного меню.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку налаштувань "гайковий ключ"; 3. Натиснути на кнопку "Назад" (стрілочки вліво).	Гравець повертається у головне меню гри після натискання на кнопку "Назад" (стрілочки вліво)	Пройдений
Беззвучний режим.	1.Вимкнути звук на пристрої; 2. Включити звук у грі; 3. Звернути увагу на відсутність звуку в грі.	Коли на пристрої ввімкнено безучний режим, а у грі перемикач звуку увімкнено, ви не почуєте звукові ефекти та музику у грі через відключений звук на пристрої.	Пройдений

Перехід до розділу "Магазин" через кнопку "+"	1. Запустити гру "XRacer"; 2. Натиснути на кнопку "+".	Гравцеві відкривається сторінка магазину після натискання на кнопку "+".	Пройдений
Перехід до розділу "Магазин" через кнопку "Store"	1. Запустити гру "XRacer"; 2. Натиснути на кнопку "Store".	Гравцю відкривається сторінка магазину після натискання на кнопку "Store".	Пройдений
Купівля № 1 (300 монет)	1. Запустити гру "XRacer"; 2. Перейти до розділу "Магазин"; 3. Натиснути на піктограму перегляду	Баланс гравця поповнюється 300 монет після перегляду реклами.	Пройдений

	реклами (кнопка play); 4. Зачекати 7 секунд реклами; 5. Натиснути на закриття екрана; 6. Звернути увагу на внутрішньоігровий баланс.		
Покупка №2 (600 монет)	1. Запустити гру "XRacer"; 2. Перейти до розділу "Магазин"; 3. Натиснути на піктограму перегляду реклами (кнопка play); 4. Зачекати 7 секунд реклами; 5. Натиснути на закриття екрана; 6. Звернути увагу на внутрішньоігровий баланс.	Баланс гравця поповнюється на 600 монет після перегляду реклами.	Пройдений
Купівля № 3 (900 монет)	1. Запустити гру "XRacer"; 2. Перейти до розділу "Магазин"; 3. Натиснути на піктограму перегляду	Баланс гравця поповнюється на 900 монет після перегляду реклами.	Пройдений

	реклами (кнопка play); 4. Зачекати 7 секунд реклами; 5. Натиснути на закриття екрана; 6. Звернути увагу на внутрішньоігровий баланс.		
Купівля № 4 (1200 монет)	1. Запустити гру "XRacer"; 2. Перейти до розділу "Магазин"; 3. Натиснути на піктограму перегляду реклами (кнопка play); 4. Зачекати 7 секунд реклами; 5. Натиснути на закриття екрана; 6. Звернути увагу на внутрішньоігровий баланс.	Баланс гравця поповнюється на 1200 монет після перегляду реклами.	Пройдений
Купівля бандла № 5 (1500 монет)	1. Запустити гру "XRacer"; 2. Перейти до розділу "Магазин"; 3. Натиснути на піктограму перегляду	Баланс гравця поповнюється на 1500 монет після перегляду реклами.	Пройдений

	реклами (кнопка play); 4. Зачекати 7 секунд реклами; 5. Натиснути на закриття екрана; 6. Звернути увагу на внутрішньоігровий баланс.		
Вихід із магазину.	1. Запустити гру "XRacer"; 2. Перейти до розділу "Магазин"; 3. Натиснути на кнопку "Назад" (стрілочки вліво)	Гравець успішно повертається в меню гри після натискання на кнопку "Назад"	Пройдений
Повторне придбання після успішного перегляду реклами	1. Запустити гру "XRacer"; 2. Перейти до розділу "Магазин"; 3. Успішно здобути будь-яку нагороду за перегляд реклами; 4. Повторно натиснути на цей же піктограму перегляду реклами.	Відображається інформер "Ads are not ready!" щоб гравець не зміг накручувати собі баланс за допомогою безперервного перегляду реклами.	Пройдений

Перехід у гараж.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку гаража (будиночок).	Гравець переходить на екран гаража.	Пройдений
Вихід із гаража.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку гаража (будиночок); 3. Натиснути на кнопку "Назад" (стрілки вліво).	Гравець успішно повертається до головного меню гри.	Пройдений
Свайп автомобілів у гаражі.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку гаража (будиночок); 3. Натиснути на кнопки вліво/вправо; 4. Звернути увагу на вітрину.	Гравець успішно гортає карусель автомобілів на вітрині.	Пройдений
Купівля автомобіля.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку гаража (будиночок); 3. Дістати до машини з кнопкою "Buy for..."; 4. Натиснути на цю кнопку.	Куплена машина успішно додається в гараж, а з рахунку гравця списується сума, що дорівнює ціні автомобіля.	Пройдений

Стайлінг автомобіля.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку гаража (будиночок); 3. Вибрати своє авто, яке хочемо стилізувати; 4. Натиснути на кнопку стайлінгу (болнчик); 5. Вибрати потрібний колір; 6. Тапнути на закриття екрана стайлінгу (стрілки вниз)	Гравець успішно застосовує обраний колір до даного автомобіля.	Пройдений
Купівля нового кольору.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку гаража (будиночок); 3. Вибрати своє авто, яке хочемо стилізувати; 4. Натиснути на кнопку стайлінгу (болнчик); 5. Вибрати потрібний платний колір;	Гравець успішно купує новий колір для автомобіля.	Пройдений

	6. Натиснути на кнопку "Buy Color".		
Детейлінг автомобіля.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку гаража (будиночок); 3. Вибрати своє авто, яке хочемо апгрейдити;4. Натиснути на кнопку дітейлінгу (гайковий ключ); 5. Натиснути на необхідну характеристику;	Гравець успішно апгрейдить обрану деталь автомобіля.	Пройдений
Фоторежим.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку гаража (будиночок); 3. Натиснути на кнопку фоторежиму (фотоапарат).	Перед гравцем залишиться тільки вітрина з автомобілем, який можна крутити і зробити гарний скріншот.	Пройдений
Вихід із фоторежиму	1. Запустити гру "XRacer"; 2. Натиснути на кнопку гаража (будиночок);	Гравець повертається до головного меню гри після повторного натискання на	Пройдений

	3. Натиснути на кнопку фоторежиму (фотоапарат); 4. Повторити п.3	кнопку фоторежиму.	
Купівля автомобіля без грошей.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку гаража (будиночок); 3. Дістати до машини з кнопкою "Buy for..."; 4. Натиснути на цю кнопку.	Гравця переводять на сторінку магазину, де він може придбати внутрішньоігрову валюту.	Пройдений
Купівля нового кольору без грошей.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку гаража (будиночок); 3. Вибрати своє авто, яке хочемо стилізувати; 4. Натиснути на кнопку стайлінгу (болнчик); 5. Вибрати потрібний платний колір; 6. Натиснути на кнопку "Buy Color".	Якщо у Гравця не вистачає коштів, йому відображається курсор, який вказує на його баланс.	Пройдений

Купівля нової деталі без грошей.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку гаража (будиночок); 3. Вибрати своє авто, яке хочемо апгрейдити; 4. Натиснути на кнопку дітейлінгу (гайковий ключ); 5. Вибрати потрібну деталь.	Якщо у Гравця не вистачає коштів, йому відображається курсор, який вказує на його баланс.	Пройдений

Перехід на екран рівнів.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку "Play".	Гравець потрапляє на екран вибору рівня.	Пройдений
Вихід із екрана рівнів.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку "Play"; 3. Натиснути на кнопку "Назад" (стрілки вліво).	Гравець успішно повертається до головного меню гри.	Пройдений
Перехід у доступний рівень.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку "Play"; 3. Натиснути на доступний рівень.	Гравець переходить до вибору карток доступного рівня.	Пройдений

Перехід на доступну карту.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку "Play"; 3. Натиснути на доступний рівень; 4. Натиснути на доступну карту.	Гра завантажує Гравцю вибрану картку.	Пройдений
Купівля закритої картки.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку "Play"; 3. Натиснути на доступний рівень; 4. Натиснути на закриту карту (іконка замку).	Якщо у Гравця достатньо монет на балансі, відбувається придбання обраної закритої картки.	Пройдений
Вихід із меню вибору карт.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку "Play"; 3. Натиснути на доступний рівень; 4. Натиснути на кнопку "Назад" (стрілки вліво).		Пройдений

Перехід у недоступний рівень.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку "Play"; 3. Натиснути на недоступний рівень (іконка замка).	Гра відображає гравцю умови для переходу на закритий рівень ---------- Гра показує карти, які є в цьому рівні без можливості перейти в них; ---------- Гра не дозволяє натиснути гравця на недоступний рівень.	Невдалий

Запуск вибраної картки.	1. Запустити гру "XRacer"; 2. Натиснути на кнопку "Play"; 3. Натиснути на доступний рівень; 4. Натиснути на доступну карту.	Гра завантажує Гравцю вибрану картку.	Пройдений
Вибір режиму "One way".	1. Запустити гру "XRacer"; 2. Натиснути на кнопку "Play"; 3. Натиснути на доступний рівень; 4. Натиснути на доступну карту; 5. Виберіть "One way".	У режимі "one way" гонка відбувається по односпрямованій трасі, де всі учасники рухаються в одному напрямку.	Пройдений

Вибір режиму "Two way".	1. Запустити гру "XRacer"; 2. Натиснути на кнопку "Play"; 3. Натиснути на доступний рівень; 4. Натиснути на доступну карту; 5. Вибрати режим "Two way"	У режимі "two way" гонка відбувається двонаправленою трасою, де учасники рухаються як проти, так і в напрямку один до одного.	Пройдений
Зміна камери.	1. Запустити будь-яку гонку; 2. Тапнути на кнопку зміни положення камери (фотоапарат).	Гравець змінює положення камери, а саме вибирає від якої особи відбуватиметься геймлей.	Пройдений
Відкриття меню паузи.	1. Запустити будь-яку гонку; 2. Натиснути на кнопку меню (пауза).	Гравець відкриває меню паузи після натискання на спеціальну іконку.	Пройдений
Керування налаштуваннями гри в меню паузи.	1. Запустити будь-яку гонку; 2. Натиснути на кнопку меню (пауза); 3. Регулювати геймлпей за допомогою налаштувань.	Кейс аналоги сценаріям налаштувань гри з головного меню.	Пройдений
Повернення до гри.	1. Запустити будь-яку гонку; 2. Натиснути на кнопку меню (пауза);	Гравець повертається на екрані гонки.	Пройдений

	3. Натиснути на кнопку "Resume".		
Перезапуск гонки.	1. Запустити будь-яку гонку; 2. Натиснути на кнопку меню (пауза); 3. Натиснути на кнопку "Restart".	Гравець перезапускає поточну гонку та починає з 0.	Пройдений
Вихід у головне меню гри.	1. Запустити будь-яку гонку; 2. Натиснути на кнопку меню (пауза); 4. Натиснути на кнопку "Menu".	Гравець повертається у меню гри після натискання на кнопку "Menu".	Пройдений
Перехід до меню паузи під час відліку до старту.	1. Запустити будь-яку гонку; 2. Натиснути на кнопку меню (пауза);	Гравець перехід у меню паузи.	Невдалий

Керування автомобілем за допомогою кнопок на екрані.	1. Запустити будь-яку гонку; 2. Натиснути на кнопку меню (пауза); 3. Вибрати ручне керування; 4. Повернутись у гонку; 5. Керувати автомобілем за допомогою газу та стрілочок.	Гравець успішно змінює траєкторію їзди автомобіля по тапу на кнопки, які є на екрані.	Пройдений

Керування автомобілем за допомогою нахилу пристрою.	1. Запустити будь-яку гонку; 2. Натиснути на кнопку меню (пауза); 3. Вибрати автоматичне керування; 4. Повернутись у гонку; 5. Керувати автомобілем за допомогою газу та повороту пристрою.	Гравець успішно змінює траєкторію їзди автомобіля за допомогою повороту пристрою.	Пройдений
Використання гальма.	1. Запустити гонку; 2. Натиснути на кнопку "Газ" (педаль справа); 3. Натиснути на кнопку "Гальмо" (педаль зліва).	Автомобіль гальмує після тапа на кнопку гальма.	Пройдений
Використання нітро-прискорювача.	1. Запустити гонку; 2. Натиснути на кнопку "Газ"; 3. Натиснути на кнопку прискорювача (балон).	Автомобіль прискорюється після тапу на нітро-прискорювач.	Пройдений
Вибір різних механік керування авто.	1. Запустити гру "XRacer"; 2. Перейти в налаштування гри;	Гравцю відображається одна і та ж механіка керування авто .	Пройдений

	3. Поставити керування на автомат; 4. Запустити гонку; 5. Перейти до меню паузі; 6. Звернути увагу на механіку керування.		

Зіткнення з ботами.	1. Запустити гонку; 2. Врізатись у чужий автомобіль.	У гравця списується 1 життя з 3 після зіткнення з чужим авто.	Пройдений
Підбір білого балону з нітро.	1. Запустити гонку; 2. Витратити все нітро; 3. Проїхати крізь балон білого нітро.	У гравця заповнюється ~ до 10% від загального запасу нітро.	Пройдений
Підбір синього балону з нітро.	1. Запустити гонку; 2. Витратити все нітро; 3. Проїхати крізь балон синього нітро.	У гравця заповнюється ~ до 50% загального запасу нітро.	Пройдений
Підбір помаранчевого балону з нітро.	1. Запустити гонку; 2. Витратити все нітро; 3. Проїхати крізь балон помаранчевого нітро.	У гравця заповнюється 100% загального запасу нітро.	Пройдений

Зіткнення з ботом під час відновлення.	1. Запустити гонку; 2. Врізатися у чужий автомобіль; 3. Поки авто відновлюється (блимає) врізатися в інший автомобіль.	Гравець проїжджає крізь чуже авто поки йде процес відновлення або має захист від зіткнення поки йде відновлення.	Невдалий

Заробляти поїнти за проїзд біля бота.	1. Запустити гонку; 2. Проїхати дуже близько до боту.	Гравцю нараховуються очки в розділі "Score".	Пройдений
Заробляти поїнти за проїзд на великій швидкості.	1. Запустити гонку; 2. Набрати велику швидкість 120км/рік +; 3. Продовжувати рух.	Гравцю нараховуються очки в розділі "Score".	Пройдений
Заробіток поїнтів за проходження дистанції без зіткнення	1. Запустити гонку; 2. Їхати багато миль без зіткнень.	Гравцю нараховуються очки в розділі "Score".	Пройдений
Заробляти поїнти за виконання завдань.	1. Запустити гонку; 2. Проходити завдання.	Гравцю нараховуються очки в розділі "Score".	Пройдений
Відображення результату після закінчення гонки.	1. Запустити гонку; 2. Пройти остаточно гонку;	Гравцю показується зведення зароблених очок за гонку.	Пройдений

	3. Звернути увагу до таблицю.		
Відображення таблиці результату після витрачання останнього життя.	1. Запустити гонку; 2. Витратити всі 3 життя; 3. Звернути увагу до таблицю.	Гравцю показується зведення зароблених очок за гонку.	Пройдений
Перехід у головне меню гри.	1. Перейти на таблицю результатів; 2. Натиснути на кнопку "Menu".	Гравець переходить у головне меню гри.	Пройдений
Рестарт гонки.	1. Перейти на таблицю результатів; 2. Натиснути на кнопку "Restart".	Гравець починає знову гонку.	Пройдений
Перехід до наступного етапу гонки.	1. Перейти на таблицю результатів; 2. Тапнути на кнопку "Next".	Игрок переходит на следующий этап гонки.	Пройдений

3.2 Тестування продуктивності

Це набір типів тестування, спрямованих на відтворення запитів користувача в системі та порівняння очікуваних результатів з отриманими показниками, а також визначення швидкості процедур, стабільності, надійності та масштабованості системи в цілому. Отримані результати дозволяють виявляти вразливості з пропускною здатністю програми, часом завантаження, обробкою великих обсягів даних та запобіганням їх використанню у додатку.

Як показує практика, найбільш поширеними проблемами продуктивності додатків є час завантаження та відгуку.

Час завантаження вимірює час запуску програми зі збільшенням навантаження чи транзакцій користувачів. Індикатори, що надійшли, як правило, не повинні перевищувати 5-10 секунди.

Час відповіді визначає час від запиту виконання певних процесів чи транзакцій до отримання відповіді.

Чим нижчий час, тим краща продуктивність. Нижче представлена таблиця з тестуванням продуктивності додатку (таб 2):

Таблиця 2 – Тест продуктивності

Назва	**Результат**
Час встановлення гри	0.26 с
Вага гри	141 MB
Швидкість запуску від початку до головного меню	9 с
Швидкість реагування всередині меню	Миттєве
Запуск реклами	Безперебійне
Затримка під час відтворення реклами	Відсутнє
Переривання звуку	Відсутнє
Підвисання звуку	Відсутнє
Проблеми зі світлом	Відсутнє
Продуктивність при кількох відкритих додатках	Безперебійне
Завантаження траси	3 с
ФПС на порожній трасі	Високий
ФПС на заселеній трасі	Високий
ФПС при використанні нітро	Середній

FPS на низьких налаштуваннях	Високий
FPS на середніх налаштуваннях	Середній
FPS на високих налаштуваннях	Середній
Текстури на карті	Завжди низької якості
Енергозберігаючий режим	На якість не впливає
FPS на картах зі снігом	Середній
FPS на картах із піском	Високий
Керування за допомогою кнопок	Плавне
Керування за допомогою нахилу пристрою	Плавне
Відгуки на сенсорне керування	Плавне
Затримка під час відкриття настройок гри	Відсутнє
Відображення частинок на карті (іскри, вогонь)	Середнє
Використання пам'яті за 10 хвилин гри	2.69 мб
Реалізація зіткнень	Програма реагує на зіткнення моментально, блокуючи на якийсь час геймплей для користувача.
Вихід із гри	1 с

Висновок щодо продуктивності:

Аналізуючи надані дані, можна дійти висновку, що гра має гарну продуктивністю загалом. Час встановлення гри був швидким, та її розмір відповідає середньому рівню.

Запуск гри до головного меню займає деякий час, але в меню реакція відбувається миттєво. Відсутність затримок під час запуску реклами, переривання звуку або підвисання звуку свідчить про плавність відтворення. Крім того, гра продемонструвала стабільну продуктивність навіть за наявності кількох відкритих програм.

Важливо, що FPS (кадри в секунду) на різних налаштуваннях графіки та різних трасах знаходиться на рівні від середнього до високого, що може забезпечити гладкість ігрового процесу.

Загальна реалізація зіткнень та управління також були оцінені як плавні та чуйні.

ВИСНОВКИ

У процесі виконання даної дипломної роботи було розроблено мобільний 3D-додаток X-Racer, що представляє гоночну гру із захоплюючим геймплеєм та реалістичною графікою. Основна мета роботи полягала у створенні повноцінного ігрового продукту, здатного надати користувачеві захоплюючий ігровий досвід.

У ході розробки були використані сучасні технології та інструменти, такі як Unity, які дозволили реалізувати фізичну модель та ігрову логіку. Завдяки цим інструментам вдалося досягти високого ступеня реалізму та деталізації навколишнього світу, а також створити захоплюючі гоночні траси та різноманітні автомобілі.

Основними функціональними можливостями програми є гонки з комп'ютерними суперниками, покращення автомобілів, купівля нових моделей, а також лідерборди та досягнення, що дозволяють гравцям змагатися та демонструвати свої результати.

В результаті тестування програми було встановлено його стабільну працездатність, відсутність критичних помилок і досить високу продуктивність. Інтерфейс користувача був розроблений з урахуванням зручності використання та інтуїтивної навігації, що забезпечує приємну взаємодію з грою.

Загалом розробка мобільного 3D-додатка X-Racer була успішною і досягла поставлених цілей.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Основи роботи в Unity 3D, (Електронний ресурс) URL:https://ec.europa.eu/programmes/erasmus-plus/project-result-content/7c7d9761-4402-467c-adab-5b09579cb8fd/2017_KNTU_Module2.pdf

2. Акселерометр у телефоні: що це таке? (Електроний ресурс) https://androidnik.ru/chto-takoe-datchik-akselerometra-v-telefone/

3. Коллайдери (Електроний ресурс) URL:https://docs.unity3d.com/2018.4/Documentation/Manual/CollidersOverview.html

4. Тестування продуктивності (Електроний ресурс) URL:https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BF%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D1%96

5. Г.М. Розробка ігор на Unity, 2018.

6. Д.Г. Бонд Unity та C#. Геймдев від ідеї до реалізації, 2020.

7. Г.Д. Ігровий двигун. Програмування та внутрішній пристрій, 2022.

8. Unity Game Development in 24 Hours, Sams Teach Yourself" by Mike Geig, 2018.

9. J.L. "Unity Virtual Reality Projects", 2015.

10. P.B-A. and J.M. "Unity Game Development Cookbook: Essentials for Every Game", 2019.

11. J.H. "Unity in Action: Multiplatform Game Development in C#", 2016

12. J.H. "Unity in Action: Multiplatform Game Development in Unity 5", 2015

ДОДАТКИ

Додаток А – Технічне завдання

ЗАТВЕРДЖЕНО
1116130.01310-01-ЛЗ

РОЗРОБКА МОБІЛЬНОГО 3D-ДОДАТКУ X-RACER.
МОДУЛІ МЕХАНІК

Технічне завдання

1116130.01310-01

Листів 11

2023

ЗМІСТ

1. ВВЕДЕННЯ

X-Racer пропонує унікальний геймплей, що поєднує в собі захоплюючі траси, реалістичну фізику та гарну графіку. Різноманітність автомобілів, 15 унікальних рівнів, можливість покращувати технічні характеристики та зовнішній вигляд автомобіля – все це буде наявне в мобільному додатку.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є наказ від 07.12.22 №1209ст ректора Українського державного університету науки і технологій “Про призначення наукових керівників та затвердження тем бакалаврських робіт” за спеціальністю 121 “Інженерія програмного забезпечення» факультету “Комп’ютерних технологій і систем” по кафедрі “Комп’ютерні інформаційні технології”.

Тема дипломної роботи - “Розробка мобільного 3D-додатку X-Racer. Модулі механіки”. Керівник - доцент Андрющенко В. О.

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення: Мобільний 3D-додаток X-Racer призначений для надання користувачам можливості грати у захоплюючі перегони на швидких автомобілях у віртуальному середовищі. Основна мета додатку - забезпечити високу якість графіки та реалістичну фізику руху автомобілів, щоб забезпечити найкращий досвід.

Функції та особливості додатку X-Racer включають в себе:

- Вибір різних моделей автомобілів із унікальними характеристиками.
- Різна місцевость для гонок.
- Можливість модернізувати автомобілі, удосконалюючи їх швидкість, керованість та гальмування.
- Ігрова валюта або монети, які можна заробити або купити, щоб витратити на удосконалення автомобілів або розблокування нових трас та автомобілів.

4. ВИМОГИ ДО ПРОГРАМИ

4.1 Вимоги до функціональних характеристик

Основні вимоги до функціональних характеристик додатку X-Racer включають:

1. Геймплей
    - Забезпечення реалістичної фізики руху автомобілів.
    - Можливість керування автомобілем за допомогою сенсорного екрану, гіроскопу або інших вхідних пристроїв/
    - Різноманітність трас.
2. Вибір автомобілів:
    - Наявність різних моделей автомобілів з унікальними характеристиками.
    - Можливість вдосконалювати автомобілі, покращуючи їх швидкість, керованісить та гальмування.
3. Управління:
    - Інтуїтивний і зручний інтерфейс користувача для керування автомобілем.
    - Варіанти керування, що задовольняють потреби користувачів (сенсорний екран та гіроскоп).
4. Прогрес гри:
    - Система валюти для витрачання на вдосконалення автомобілів або розблокування нових елементів гри.

4.2 Вимоги до надійності

Додаток повинен працювати без збоїв та несправностей. Він повинен бути стійким до помилок та некоректної поведінки, щоб забезпечити безперебійну

геймплейну сесію для користувачів.

4.3 Умови експлуатації

Програмний продукт повинен використовуватись у приміщеннях, які відповідають умовам роботи ЕОМ, а саме мають такі кліматичні, санітарні та гігієнічні умови, які відповідають ДНАОП 0.00-1.31-99 (див. табл. 1).

Таблиця 3.1 — Кліматичні умови

Пора року	Категорія категорія робіт згідно з ГОСТ 12.1-005-88	Температура повітря, град.С	Відносна вологість повітря, %	Швидкість руху повітря, м/с
		оптимальна	оптимальна	оптимальна
Холодна	легка-1 а	22-24	40-60	0,1
	легка-1 б	21-23	40-60	0,1
Тепла	легка-1 а	23-25	40-60	0,1
	легка-1 б	22-24	40-60	0,2

Працювати з програмою може людина, що має навички роботи з персональним комп’ютером та мобільними пристроями.

4.4 Вимоги до складу та параметрів технічних засобів

Склад технічних засобів: мобільний пристрій з версією не меньше 7 Android.

4.5 Вимоги до інформаційної та програмної сумісності

Програмний продукт розробляється тільки для платформи Android.

4.6 Вимоги до маркування і упаковки

Упаковка програмного продукту, включаючи документацію повинна бути захищена від пошкоджень різного роду (механічних, кліматичних). На упаковці повинно бути вказана назва продукту, номер версії (якщо вона змінювалась), мінімальні системні вимоги. На зворотній стороні упаковки вказується

розробник та його юридична адреса.

4.7 Вимоги до транспортування і зберігання

Транспортування повинне забезпечувати збереження програмного продукту, його цілісність і запобігання несанкціонованого доступу до нього.

5. ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу програмної документації мають входити:

- специфікація;
- текст програми;

Програмна документація повинна відповідати вимогам ДСТУ [1].

6. СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Таблиця 5.1 – Стадії та етапи розробки

№	азва етапів кваліфікаційної роботи	Строк виконання етапів	Примітка
1	Вступ	03.04.2023 – 05.04.2023	
2	Огляд літератури Unity, фізика 3D-об`єктів	06.04.2023 – 09.04.2023	
3	Проєктування основних модулів механіки	10.04.2023 – 24.04.2023	
4	Постановка завдання	25.04.2023 – 02.05.2023	
5	Розробка та програмування логіки механік	03.05.2023 – 10.05.2023	30%
6	Розробка та програмування додаткового функціоналу	11.05.2023 – 18.05.2023	
7	Тестування	19.05.2023 – 26.05.2023	60%
8	Оформлення пояснювальної записки	27.05.2023 – 02.06.2023	
9	Розробка демонстраційних матеріалів	03.06.2023 – 18.06.2023	100%
10	Подання кваліфікаційної роботи до кафедри	19.06.2023 – 26.06.2023	
11	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	27.06.2023	

7. ПОРЯДОК КОНТРОЛЮ І ПРИЙМАННЯ

Контроль виконання здійснює керівник розробки Андрющенко В.А. Прийом здійснюється уповноваженою комісією.

8. БІБЛІОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем: методичні вказівки до дипломного проектування та лабораторних робіт/уклад.: Ю.М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. - 38 с.

Додаток Б – Специфікація

ЗАТВЕРДЖЕНО
1116130.01310-01-ЛЗ

РОЗРОБКА МОБІЛЬНОГО 3D-ДОДАТКУ X-RACER. МОДУЛІ МЕХАНІКИ

Специфікація

1116130.01310-01

Листів 2

2023

Специфікації

Таблиця 5.2 – Специфікації

Позначення	Найменування	Примітка
	Документація	
1116130.01310-01-ЛЗ	Лист затвердження	
1116130.01310-01	Технічне завдання	
1116130.01310-01-ЛЗ	Лист затвердження	
1116130.01310-01	Специфікація	
1116130.01310-01 12 01-ЛЗ	Лист затвердження	
1116130.01310-01 12 01	Текст програми	

Додаток В – Листи затвердження

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського
державного університету
науки і технологій
_____ Анатолій РАДКЕВИЧ
07.12.2022

РОЗРОБКА МОБІЛЬНОГО 3D-ДОДАТКУ X-RACER. МОДУЛІ МЕХАНІК

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01310-01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
_____ Вадим ГОРЯЧКІН
07.12.22
Керівник розробки
_____ Вадим АНДРЮЩЕНКО
07.12.22
Виконавець
_____ Дмитро П`ЯНИЙ
07.12.22
Нормконтролер
_____ Світлана ВОЛКОВА
07.12.22

2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського
державного університету
науки і технологій
_____ Анатолій РАДКЕВИЧ
07.12.2022

РОЗРОБКА МОБІЛЬНОГО 3D-ДОДАТКУ X-RACER. МОДУЛІ МЕХАНІК

Специфікація
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01310-01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
_____ Вадим ГОРЯЧКІН
07.12.22
Керівник розробки
_____ Вадим АНДРЮЩЕНКО
07.12.22
Виконавець
_____ Дмитро П`ЯНИЙ
07.12.22
Нормконтролер
_____ Світлана ВОЛКОВА
07.12.22

2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського
державного університету
науки і технологій
_____ Анатолій РАДКЕВИЧ
07.12.2022

РОЗРОБКА МОБІЛЬНОГО 3D-ДОДАТКУ X-RACER. МОДУЛІ МЕХАНІКИ

Текст програми
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01310-01 12 01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
_____ Вадим ГОРЯЧКІН
07.12.22
Керівник розробки
_____ Вадим АНДРЮЩЕНКО
07.12.22
Виконавець
_____ Дмитро П`ЯНИЙ
07.12.22
Нормконтролер
_____ Світлана ВОЛКОВА
07.12.22

2023

Додаток Г – Текст програми

ЗАТВЕРДЖЕНО

1116130.01310-01 12 01-ЛЗ

РОЗРОБКА МОБІЛЬНОГО 3D-ДОДАТКУ X-RACER. МОДУЛІ МЕХАНІКИ

Текст програми

1116130.01310-01 12 01

Листів 46

2023

HR_Controller_Type

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
public class HR_Controller_Type : MonoBehaviour {

	public controllerType _controllerType;
	public enum controllerType{ keypad, accelerometer
,buttons}
	private Button sprite;
	private Color defCol;
	private Button Gas;
	public Image[] controllerImgaes;
	public Sprite[] yellowGreen;
	void Start () {

		sprite = GetComponent<Button>();
//		defCol = sprite.image.color;
		if (!SceneManager.GetSceneByName ("Main
Menu").isLoaded && !SceneManager.GetSceneByName ("new
main").isLoaded) {
			Gas = GameObject.Find
("Gas").GetComponent<Button> ();
		}
		if(!PlayerPrefs.HasKey("ControllerType"))

	PlayerPrefs.SetInt("ControllerType", 0);
		int a = PlayerPrefs.GetInt ("AutoRace");
//		int a = PlayerPrefs.GetInt ("Control");
		switch (a) {
		case 1:
			first ();
			break;
		case 2:
			third ();
			break;
		default :
			second ();
			break;
		}
		Check ();

	}

	public void OnClick () {

		General.Instance.playClickSound ();
		if(_controllerType == controllerType.keypad){
			first ();
		}

		if(_controllerType ==
controllerType.accelerometer){
			second ();
		}
		if(_controllerType == controllerType.buttons){
			third ();
		}

		HR_Controller_Type[] ct =
GameObject.FindObjectsOfType<HR_Controller_Type>();

		foreach(HR_Controller_Type cts in ct){
			cts.Check();
		}
	}

	void first(){

		PlayerPrefs.SetInt("ControllerType", 1);
		PlayerPrefs.SetInt ("AutoRace",1);
		if (!SceneManager.GetSceneByName ("Main
Menu").isLoaded && !SceneManager.GetSceneByName ("new
main").isLoaded) {
			Gas.interactable = false;
			Gas.GetComponent<Image>
().color = new Color (0f, 0f, 0f, 0f);

	Gas.GetComponent<RCC_UIController> ().pressing =
true;
			controllerImgaes [0].sprite =
yellowGreen [1]; //green

	controllerImgaes[1].sprite=yellowGreen[0]; //yellow

	controllerImgaes[2].sprite=yellowGreen[0]; //yellow
		} else {
			controllerImgaes [0].color = new
Color (1, 1, 0, 1);
			controllerImgaes [1].color = new
Color (0.3f, 0.3f, 0.3f, 0.8f);
			controllerImgaes [2].color = new
Color (0.3f, 0.3f, 0.3f, 0.8f);
		}
	}
	void second()
	{
		PlayerPrefs.SetInt("ControllerType", 1);
		PlayerPrefs.SetInt ("AutoRace",0);
		if (!SceneManager.GetSceneByName ("Main
Menu").isLoaded && !SceneManager.GetSceneByName ("new
main").isLoaded) {
			Gas.interactable = true;
			Gas.GetComponent<Image>
().color = new Color (1f, 1f, 1f, 1f);

	Gas.GetComponent<RCC_UIController>().pressing =
false;
			controllerImgaes [0].sprite =
yellowGreen [0];

	controllerImgaes[1].sprite=yellowGreen[1];

	controllerImgaes[2].sprite=yellowGreen[0];
		}else {
			controllerImgaes [1].color = new
Color (1, 1, 0, 1);
			controllerImgaes [0].color = new
Color (0.3f, 0.3f, 0.3f, 0.8f);
			controllerImgaes [2].color = new
Color (0.3f, 0.3f, 0.3f, 0.8f);
		}
	}
	void third()
	{
		PlayerPrefs.SetInt("ControllerType", 0);
		PlayerPrefs.SetInt ("AutoRace",2);
		if (!SceneManager.GetSceneByName ("Main
Menu").isLoaded && !SceneManager.GetSceneByName ("new
main").isLoaded) {
			Gas.interactable = true;
			Gas.GetComponent<Image>
().color = new Color (1f, 1f, 1f, 1f);
```

```
            Gas.GetComponent<RCC_UIController>().pressing =
false;
                                    controllerImgaes [0].sprite =
yellowGreen [0];

            controllerImgaes[1].sprite=yellowGreen[0];

            controllerImgaes[2].sprite=yellowGreen[1];
                        }else {
                                    controllerImgaes [2].color = new
Color (1, 1, 0, 1);
                                    controllerImgaes [0].color = new
Color (0.3f, 0.3f, 0.3f, 0.8f);
                                    controllerImgaes [1].color = new
Color (0.3f, 0.3f, 0.3f, 0.8f);
                        }
            }
            void Check(){

                        if(PlayerPrefs.GetInt("ControllerType") == 0){
                                    if(_controllerType ==
controllerType.keypad){
//                                          sprite.image.color = new
Color(.667f, 1f, 0f);
                                    }
                                    if(_controllerType ==
controllerType.accelerometer){
//                                          sprite.image.color =
defCol;
                                    }
                        }
                        if(PlayerPrefs.GetInt("ControllerType") == 1){
                                    if(_controllerType ==
controllerType.keypad){
//                                          sprite.image.color =
defCol;
                                    }
                                    if(_controllerType ==
controllerType.accelerometer){
//                                          sprite.image.color = new
Color(.667f, 1f, 0f);
                                    }
                        }


            if(GameObject.FindObjectOfType<RCC_CarControllerV
3>())

            GameObject.FindObjectOfType<RCC_CarControllerV3>
().GetControllerType();

            }

}
```

HR_GamePlayHandler

```
using UnityEngine;
using UnityEngine.UI;
using System;
using System.Collections;
using UnityEngine.SceneManagement;
using DG.Tweening;
using UnityEngine.Analytics;
using GoogleMobileAds.Api;
public class HR_GamePlayHandler : MonoBehaviour {

            public AdInterstatial _adInterstatial;
            public GameObject[] playerCars;
            public GameObject playerCar;
            public GameObject countDown;
            public AudioSource completeSound;
            public AdBanner _adBanner;
            HR_PlayerHandler _handler;
            #region SINGLETON PATTERN
            public static HR_GamePlayHandler _instance;
            public static HR_GamePlayHandler Instance
            {
                        get
                        {
                                    if (_instance == null){
                                                _instance =
GameObject.FindObjectOfType<HR_GamePlayHandler>();
                                    }

                                    return _instance;
                        }
            }
            #endregion
            [Header("Time Of The Scene")]
            public DayOrNight dayOrNight;
            public enum DayOrNight{Day, Night, Rainy, Snowy}

            [Header("Current Mode")]
            internal Mode mode;
            internal enum Mode {OneWay, TwoWay, TimeAttack,
Bomb}

            [Header("UI Canvases For GamePlay and GameOver")]
            public Canvas gameplayCanvas;
            public Canvas gameoverCanvas;
            public GameObject[] gameOverAnimBtns;
            public GameObject completeDialog;

            public GameObject camera;

            [Header("Spawn Location Of The Cars")]
            public Transform spawnLocation;

            public  GameObject player;

            private int selectedCarIndex = 0;
            private int selectedModeIndex = 0;
            private float minimumSpeed = 20f;

            internal bool gameStarted = false;
            internal bool paused = false;

            public int totalPlayedCount = 0;

            public delegate void onPlayerSpawned(GameObject p);
            public static event onPlayerSpawned OnPlayerSpawned;

            public delegate void onPlayerEnabled();
            public static event onPlayerEnabled OnPlayerEnabled;

            public delegate void onPlayerDisabled();
            public static event onPlayerDisabled OnPlayerDisabled;

            public delegate void onPlayerCrashed();
            public static event onPlayerCrashed OnPlayerCrashed;

            public delegate void onPaused();
            public static event onPaused OnPaused;

            public delegate void onResumed();
            public static event onResumed OnResumed;
            Image NosImage;
            CarController carController;
      bool SpeedRefresh = true;
      public static bool ShareButtons = false,isCheckPoint;
            public static int highwayNumber;
            public GameObject startDemoPanel,levelsChallange;
            public int checkpointIndex;
```

```
	public LevelsData levelDescription;
	Coroutine highwayStatsRoutine=null;

  void Awake(){

	  SpawnCar();
		Time.timeScale = 1;
		Scriptt.levelDone = false;
		if (MenuScript.day) {
			dayOrNight = DayOrNight.Day;
		}
		else {
			dayOrNight = DayOrNight.Night;
		}

	if(HR_HighwayRacerProperties.Instance.gameplayClips
!= null &&
HR_HighwayRacerProperties.Instance.gameplayClips.Length > 0)

	RCC_CreateAudioSource.NewAudioSource(gameObject,
"GamePlay Soundtrack", 0f, 0f, .35f,
HR_HighwayRacerProperties.Instance.gameplayClips[UnityEngine.
Random.Range(0,
HR_HighwayRacerProperties.Instance.gameplayClips.Length)], true,
true, false);

		selectedCarIndex =
PlayerPrefs.GetInt("SelectedPlayerCarIndex",0);
		selectedModeIndex =
PlayerPrefs.GetInt("SelectedModeIndex",0);
		totalPlayedCount =
PlayerPrefs.GetInt("TotalPlayedCount",0);

		switch(selectedModeIndex){
		case 0:
			mode = Mode.OneWay;
			break;
		case 1:
			mode = Mode.TwoWay;
			break;
		case 2:
			mode = Mode.TimeAttack;
			break;
		case 3:
			mode = Mode.Bomb;
			break;
		}
//		highwayNumber = 2;
		if(startDemoPanel)
			startDemoPanel.SetActive (true);
		if (highwayNumber > 0) {
			isCheckPoint = true;
			levelsChallange.SetActive (false);
		} else {
			levelsChallange.SetActive (true);
			if (LevelChallange._level == 9) {
				isCheckPoint = false; //
me change
			}
			else {
				isCheckPoint = false;
			}
		}
	}
	public void reSetTrafficMode()
	{
		selectedModeIndex =
PlayerPrefs.GetInt("SelectedModeIndex",0);
		switch(selectedModeIndex){
		case 0:
			mode = Mode.OneWay;
			break;
		case 1:
			mode = Mode.TwoWay;
			break;
		case 2:
			mode = Mode.TimeAttack;
			break;
		case 3:
			mode = Mode.Bomb;
			break;
		}
	}
	void Start () {
		gameplayCanvas.enabled = true;
		gameoverCanvas.enabled = false;
		Screen.sleepTimeout =
SleepTimeout.NeverSleep;


		StartCoroutine(WaitForGameStart());
		if(GameObject.FindGameObjectWithTag
("NosImage"))
			NosImage =
GameObject.FindGameObjectWithTag
("NosImage").GetComponent<Image> ();
		CollectableControl.isShield = false;
    ShareButtons = false;
		checkpointIndex = 1;
  }

	IEnumerator WaitForGameStart(){
		yield return new WaitForSeconds(6);
		gameStarted = true;
		startDemoPanel.SetActive (false);
    CarController.GameSatart = true;
    //CarController.canControl = true;
   //CarController.ins.rb.velocity = new Vector3(0, 0, 40);
  }
	IEnumerator setHighWayPlayerpref()
	{
			yield return new WaitForSeconds
(1f);
			if (PlayerPrefs.GetFloat
("HighWay" + highwayNumber + "Stats") < _handler.distance) {
				if (PlayerPrefs.GetFloat
("HighWay" + highwayNumber + "Stats") <
levelDescription.highwayDistance [highwayNumber - 1]) {

	PlayerPrefs.SetFloat (("HighWay" + highwayNumber +
"Stats"), _handler.distance);
				}

			}
			if (highwayStatsRoutine != null)
				highwayStatsRoutine =
StartCoroutine (setHighWayPlayerpref ());
	}
	public bool checkHighwayDistanceComplete()
	{
		if (highwayNumber > 0) {
			if (PlayerPrefs.GetFloat
("HighWay" + highwayNumber + "Stats") >=
levelDescription.highwayDistance [highwayNumber - 1]) {
				stopCar ();

	General.Instance.highwayCompleted ();
				StopCoroutine
(highwayStatsRoutine);
				highwayStatsRoutine =
null;
				PlayerPrefs.SetFloat
(("HighWay" + highwayNumber +
"Stats"),(levelDescription.highwayDistance [highwayNumber - 1]));
```

```
            if(!PlayerPrefs.HasKey("HighwayAchievement"+highwa
yNumber))
                                                {

            if(highwayNumber==1)

            General.Instance.callAchievement (16); //highway
achievement starts from 16-19

            PlayerPrefs.SetInt("HighwayAchievement"+highwayNum
ber,1);
                                                }
                                                return true;
                                    }
                        }
                        return false;
            }
            public void stopCar()
            {
                        carController.stopOnComplete ();
      if
(player.GetComponent<CarStats>().shieldParticle.activeInHierarchy)

player.GetComponent<CarStats>().shieldParticle.SetActive(false);
      if
(player.GetComponent<CarStats>().shieldTruck.activeInHierarchy)
      {

player.GetComponent<CarStats>().shieldTruck.transform.GetChild(
0).gameObject.SetActive(false);

player.GetComponent<CarStats>().shieldTruck.transform.GetChild(
2).gameObject.SetActive(false);

player.GetComponent<CarStats>().shieldTruck.transform.GetChild(
3).gameObject.SetActive(false);
      }
   }
            public void countDownAnim()
            {
                        countDown.SetActive (false);
                        countDown.SetActive (true);
            }
   public void SpawnCar () {

                        if (mode != Mode.Bomb) {
                                    player = (GameObject)Instantiate
(playerCars [MenuScript.selectedCar]);
                                    player.SetActive (false);
                        }
                        player.transform.position =
spawnLocation.transform.position;
                        OnPlayerSpawned (player);
                        player.transform.rotation =
Quaternion.identity;
                        carController =
player.GetComponent<CarController> ();
                        player.SetActive (true);
                        player.GetComponent<CarStats>
().setCarControllerValues ();
                        SmoothFollow.target=player.transform;
                        SmoothFollow.hoodCam =
player.transform.GetChild(0).transform;
                        _handler =
player.GetComponent<HR_PlayerHandler> ();
                        if (highwayNumber > 0) {
                                    highwayStatsRoutine =
StartCoroutine (setHighWayPlayerpref ());
                        }
            }

            public IEnumerator OnGameOver(float delayTime){
                        _adBanner.DisableBanner();
                        _adInterstatial.ShowAd();
                        if (General.levelComplete) {
                                    delayTime = 3f;
      Time.timeScale = 1.0f;
      Time.fixedDeltaTime = 0.0167f;
      yield return new WaitForSeconds(2);
      completeDialog.SetActive (true);
                        }
                        else {
                                    delayTime = 2f;
                        }
                        yield return new WaitForSeconds(delayTime);
                        completeDialog.SetActive (false);
                        gameplayCanvas.enabled = false;
                        OnPaused ();
                        gameoverCanvas.enabled = true;
                        //foreach (GameObject btn in
gameOverAnimBtns) {
                        //
            btn.GetComponent<DOTweenAnimation> ().DORestart
();
                        //}
      ShareButtons = true;

            GameObject.FindObjectOfType<HR_Scoreboard>().Disp
layResults();
                        switch(mode){

                        case Mode.OneWay:

            PlayerPrefs.SetInt("bestScoreOneWay",
(int)player.GetComponent<HR_PlayerHandler>().score);
                                    break;
                        case Mode.TwoWay:

            PlayerPrefs.SetInt("bestScoreTwoWay",
(int)player.GetComponent<HR_PlayerHandler>().score);
                                    break;
                        case Mode.TimeAttack:

            PlayerPrefs.SetInt("bestScoreTimeAttack",
(int)player.GetComponent<HR_PlayerHandler>().score);
                                    break;
                        case Mode.Bomb:

            PlayerPrefs.SetInt("bestScoreBomb",
(int)player.GetComponent<HR_PlayerHandler>().score);
                                    break;


                        }

                        totalPlayedCount++;
                        PlayerPrefs.SetInt("TotalPlayedCount",
totalPlayedCount);
            }

            void OnLevelWasLoaded(){
                        Time.timeScale = 1;
                        AudioListener.pause = false;
            }

            public void MainMenu()
            {
                        if (Scriptt.levelDone)
                        {

            SceneManager.LoadScene(SceneManager.GetActiveScen
e().buildIndex);

                        }
```

```
                else
                {
                        LoadingScript.leveltoload = 1;
                        General.NextLevelAd = true;

                        Application.LoadLevel (8);
                }
        }

        public void RestartGame(){
//      #if !UNITY_EDITOR
//              if (Ads_Manager.Instance) {
//                      if (PlayerPrefs.GetInt
("ADSUNLOCK") != 1) {
//
        Ads_Manager.Instance.HideLargeAdmobBanner ();
//
        Ads_Manager.Instance.HideLargeAdmobBanner1 ();
//                      }
//              }
//#endif
                if (Scriptt.levelDone) { //next level
                        LoadingScript.leveltoload = 1;
                        General.NextLevelAd = true;
                        Application.LoadLevel (1);
                } else { //restart level

        SceneManager.LoadScene(SceneManager.GetActiveScen
e().buildIndex);

                }


        }
        public void nextLevel()
        {

                if (LevelChallange.instance) {
                        if (LevelChallange._level <
MenuScript.totalLevels) {
                                LevelChallange._level
+= 1;

        SceneManager.LoadScene
(SceneManager.GetActiveScene ().buildIndex);
                        }
                        else {

        LoadingScript.leveltoload = 1;
                                General.NextLevelAd =
true;
                                Application.LoadLevel
(1);
                        }
                }
//#if !UNITY_EDITOR
//              if (Ads_Manager.Instance) {
//                      if (PlayerPrefs.GetInt
("ADSUNLOCK") != 1) {
//
        Ads_Manager.Instance.HideLargeAdmobBanner ();
//
        Ads_Manager.Instance.HideLargeAdmobBanner1 ();
//                      }
//              }
//#endif
    }
    public void RestartGame2()
    {
                if (PlayerPrefs.GetInt ("ADSUNLOCK") != 1)
{
//#if !UNITY_EDITOR
//
        Ads_Manager.Instance.HideLargeAdmobBanner ();
//
        Ads_Manager.Instance.HideLargeAdmobBanner1 ();
//#endif
                }
        if (Scriptt.levelDone) //next level
        {
           LoadingScript.leveltoload = 1;
           General.NextLevelAd = true;
           Application.LoadLevel(1);

        }
        else //restart level
        {
          // SceneManager.LoadScene("11-Levels");

        }

    }


        public void GotoLevelSelection()
        {
                if (PlayerPrefs.GetInt ("ADSUNLOCK") != 1)
{
//#if !UNITY_EDITOR
//
        Ads_Manager.Instance.HideLargeAdmobBanner ();
//
        Ads_Manager.Instance.HideLargeAdmobBanner1 ();
//#endif
                }

                LoadingScript.leveltoload = 1;
                General.NextLevelAd = true;
                Application.LoadLevel(1);

                PlayerPrefs.SetInt("GotoLevels",1);
        }


        public void Paused(){

                paused = !paused;

                if(paused)
                        OnPaused ();
                else
                        OnResumed ();

        }

#region free cash
        public GameObject freeCashPanel;
        public void openCashPanel()
        {
                freeCashPanel.SetActive (true);
        }
        public void closeCashPanel()
        {
                General.Instance.playClickSound ();
                freeCashPanel.SetActive (false);
        }
#endregion

}
```

HR_HighwayRacerProperties

```
using UnityEngine;
```

```
using System.Collections;

[System.Serializable]
public class HR_HighwayRacerProperties : ScriptableObject {

	public static HR_HighwayRacerProperties instance;
	public static HR_HighwayRacerProperties Instance
	{
		get
		{
			if(instance == null)
				instance =
Resources.Load("HR_Assets/HR_HighwayRacerProperties") as
HR_HighwayRacerProperties;
			return instance;
		}

	}

	public int _minimumSpeedForGainScore;
	public int _minimumSpeedForHighSpeed;
	public int _minimumCollisionForGameOver;

	public Color _defaultBodyColor;
	public bool _defaultBloom;
	public bool _defaultHQLights;
	public bool _defaultFlares;
	public bool _shakeCamera;
	public int _totalDistanceMoneyMP;
	public int _totalNearMissMoneyMP;
	public int _totalOverspeedMoneyMP;
	public int _totalOppositeDirectionMP;

	public int toolbarSelectedIndex;
	public bool _1MMoneyForTesting;

	public GameObject[] selectablePlayerCars;
	public GameObject[] upgradableWheels;
	public GameObject lightProjector;
	public GameObject tailLightTrails;
	public GameObject attachableSiren;
	public GameObject explosionEffect;

	public GameObject exhaustGas;

	public AudioClip[] mainMenuClips;
	public AudioClip[] gameplayClips;
	public AudioClip buttonClickAudioClip;
	public AudioClip nearMissAudioClip;
	public AudioClip labelSlideAudioClip;
	public AudioClip countingPointsAudioClip;
	public AudioClip bombTimerAudioClip;
	public AudioClip sirenAudioClip;

	public LayerMask trafficCarsLayer;

}
```

HR_MainMenuHandler

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
using System;
using UnityEngine.SceneManagement;

public class HR_MainMenuHandler : MonoBehaviour {

	[Header("Spawn Location Of The Cars")]
	public Transform carSpawnLocation;

	internal GameObject[] cars;
	internal int carIndex = 0;
	internal int modeIndex = 0;

	[Header("UI Menus")]
	public GameObject optionsMenu;
	public GameObject carSelectionMenu;
	public GameObject modsSelectionMenu;
	public GameObject sceneSelectionMenu;
	public GameObject coinsMenu;
	public GameObject creditsMenu;

	[Header("UI Loading Section")]
	private AsyncOperation async;
	public GameObject loadingScreen;
	public Slider loadingBar;

	internal RCC_CarControllerV3[] currentCarControllers;
	internal HR_ModApplier[] currentModAppliers;

	[Header("Other UIs")]
	public Text currency;
	public GameObject buyCarButton;
	public GameObject selectCarButton;
	public GameObject modCarPanel;
	public GameObject bestScoreOneWay;
	public GameObject bestScoreTwoWay;
	public GameObject bestScoreTimeLeft;
	public GameObject bestScoreBomb;

	void Awake(){

		Time.timeScale = 1;


	if(HR_HighwayRacerProperties.Instance.mainMenuClips
!= null &&
HR_HighwayRacerProperties.Instance.mainMenuClips.Length > 0)

	RCC_CreateAudioSource.NewAudioSource(gameObject,
"Main Menu Soundtrack", 0f, 0f, 1f,
HR_HighwayRacerProperties.Instance.mainMenuClips[UnityEngine
.Random.Range(0,
HR_HighwayRacerProperties.Instance.mainMenuClips.Length)],
true, true, false);


	if(HR_HighwayRacerProperties.Instance._1MMoneyForT
esting)
			PlayerPrefs.SetInt("Currency",
1000000);

		carIndex =
PlayerPrefs.GetInt("SelectedPlayerCarIndex", 0);

		CreateCars();
		SpawnCar();

	}



	void Update(){

		currency.text = PlayerPrefs.GetInt("Currency",
0).ToString("F0");
		if(async != null &&  !async.isDone){
			loadingBar.value = async.progress;
		}

	}

	void CreateCars(){
```

```
                cars = new GameObject[HR_PlayerCars.Instance.cars.Length];     Debug.Log ("CreateCars//////////"+cars.Length);
                currentCarControllers = new RCC_CarControllerV3[cars.Length];
                currentModAppliers = new HR_ModApplier[cars.Length];

                for(int i = 0; i < cars.Length; i++){

                        cars[i] = (GameObject)Instantiate(HR_PlayerCars.Instance.cars[i].playerCar);

        cars[i].GetComponent<RCC_CarControllerV3>().canControl = false;

        cars[i].GetComponent<RCC_CarControllerV3>().engineRunning = false;
                        cars[i].SetActive(false);
                        currentCarControllers[i] = cars[i].GetComponent<RCC_CarControllerV3>();
                        currentModAppliers[i] = cars[i].GetComponent<HR_ModApplier>();

                }

        }

        public void SpawnCar () {

//
        if(HR_PlayerCars.Instance.cars[carIndex].price <= 0 || HR_PlayerCars.Instance.cars[carIndex].unlocked)
//
        PlayerPrefs.SetInt(cars[carIndex].name + "Owned", 1);
//
//              if(PlayerPrefs.GetInt(cars[carIndex].name + "Owned") == 1){
//
        if(buyCarButton.GetComponentInChildren<Text>())
//
        buyCarButton.GetComponentInChildren<Text>().text = "";
//                      buyCarButton.SetActive(false);
//                      selectCarButton.SetActive(true);
//                      modCarPanel.SetActive(true);
//              }else{
//                      selectCarButton.SetActive(false);
//                      buyCarButton.SetActive(true);
//                      modCarPanel.SetActive(false);
//
        if(buyCarButton.GetComponentInChildren<Text>())
//
        buyCarButton.GetComponentInChildren<Text>().text = "BUY FOR\n" + HR_PlayerCars.Instance.cars[carIndex].price.ToString("F0");
//              }

//              for(int i = 0; i < cars.Length; i++){
//                      cars[i].SetActive(false);
////                    cars[i].transform.position = carSpawnLocation.position;
////                    cars[i].transform.rotation = carSpawnLocation.rotation;
//              }

//              cars[carIndex].SetActive(true);

        }

        public void BuyCar (){

                if(PlayerPrefs.GetInt("Currency") >= HR_PlayerCars.Instance.cars[carIndex].price)
                        PlayerPrefs.SetInt("Currency", PlayerPrefs.GetInt("Currency") - HR_PlayerCars.Instance.cars[carIndex].price);
                else
                        return;

                PlayerPrefs.SetInt(cars[carIndex].name + "Owned", 1);

                SpawnCar();

        }

        public void SelectCar (){

                PlayerPrefs.SetInt("SelectedPlayerCarIndex", carIndex);

        }

        public void PositiveCarIndex (){

                carIndex ++;

                if(carIndex >= cars.Length)
                        carIndex = 0;

                SpawnCar();

        }

        public void NegativeCarIndex (){

                carIndex --;

                if(carIndex < 0)
                        carIndex = cars.Length - 1;

                SpawnCar();

        }

        public void EnableMenu(GameObject activeMenu){

                if(activeMenu.activeSelf){
                        activeMenu.SetActive(false);
                        return;
                }

                optionsMenu.SetActive(false);
                carSelectionMenu.SetActive(false);
                modsSelectionMenu.SetActive(false);
                sceneSelectionMenu.SetActive(false);
                coinsMenu.SetActive(false);
                creditsMenu.SetActive(false);
                loadingScreen.SetActive(false);

                activeMenu.SetActive(true);

                if(activeMenu == modsSelectionMenu)
                        BestScores();

        }

        public void SelectScene(int levelIndex){

                SelectCar();
                EnableMenu(loadingScreen);
                async =
```

```
SceneManager.LoadSceneAsync(levelIndex);

	}

	public void SelectMode(int _modeIndex){

		PlayerPrefs.SetInt("SelectedModeIndex",
_modeIndex);
		EnableMenu(sceneSelectionMenu);

	}

	public void BestScores(){


	bestScoreOneWay.GetComponentInChildren<Text>().tex
t = "BEST SCORE\n" + PlayerPrefs.GetInt("bestScoreOneWay", 0);

	bestScoreTwoWay.GetComponentInChildren<Text>().tex
t = "BEST SCORE\n" + PlayerPrefs.GetInt("bestScoreTwoWay", 0);

	bestScoreTimeLeft.GetComponentInChildren<Text>().tex
t = "BEST SCORE\n" + PlayerPrefs.GetInt("bestScoreTimeAttack",
0);

	bestScoreBomb.GetComponentInChildren<Text>().text =
"BEST SCORE\n" + PlayerPrefs.GetInt("bestScoreBomb", 0);

	}

	public void QuitGame(){

		Application.Quit();

	}

}
```

HR_ModHandler

```
//Modification Script For Checking/Applying User Modifications.
Also Controls UI Buttons, Sliders and Texts About Modding

using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class HR_ModHandler : MonoBehaviour {

	//Classes
	private HR_MainMenuHandler menuHandler;
	private RCC_CarControllerV3 currentCar;
	private HR_ModApplier currentApplier;

	//UI Panels
	[Header("Modify Panels")]
	public GameObject colorClass;
	public GameObject wheelClass;
	public GameObject upgradesClass;

	//UI Buttons
	[Header("Modify Buttons")]
	public Button bodyPaintButton;
	public Button rimButton;
	public Button upgradeButton;
	private Color orgButtonColor;

	//UI Sliders
	[Header("Power Bars")]
	public Slider speedBar;
	public Slider handlingBar;
	public Slider brakeBar;

	[Header("Maximum Power Bars")]
	public Slider maxSpeedBar;
	public Slider maxHandlingBar;
	public Slider maxBrakeBar;

	//UI Texts
	[Header("Upgrade Levels Texts")]
	public Text speedUpgradeLevel;
	public Text handlingUpgradeLevel;
	public Text brakeUpgradeLevel;
	public Text sirenUpgradeLevel;

	void Awake () {

		orgButtonColor =
bodyPaintButton.image.color;
		menuHandler =
GameObject.FindObjectOfType<HR_MainMenuHandler>();

	}

	void Update(){

		currentCar =
menuHandler.currentCarControllers[menuHandler.carIndex];
		currentApplier =
menuHandler.currentModAppliers[menuHandler.carIndex];

		speedBar.value = Mathf.Lerp(speedBar.value,
currentCar.maxspeed / 350f, Time.deltaTime * 5f);
		handlingBar.value =
Mathf.Lerp(handlingBar.value, currentCar.highspeedsteerAngle /
22f, Time.deltaTime * 5f);
		brakeBar.value = Mathf.Lerp(brakeBar.value,
currentCar.brake / 10000f, Time.deltaTime * 5f);

		maxSpeedBar.value =
Mathf.Lerp(maxSpeedBar.value, currentApplier.maxUpgradeSpeed /
350f, Time.deltaTime * 5f);
		maxHandlingBar.value =
Mathf.Lerp(maxHandlingBar.value,
currentApplier.maxUpgradeHandling / 22f, Time.deltaTime * 5f);
		maxBrakeBar.value =
Mathf.Lerp(maxBrakeBar.value, currentApplier.maxUpgradeBrake /
10000f, Time.deltaTime * 5f);

		speedUpgradeLevel.text =
currentApplier.speedLevel.ToString("F0");
		handlingUpgradeLevel.text =
currentApplier.handlingLevel.ToString("F0");
		brakeUpgradeLevel.text =
currentApplier.brakeLevel.ToString("F0");
		sirenUpgradeLevel.text =
currentApplier.isSirenPurchased &&
currentApplier.attachedFrontSiren.activeSelf ? "ON" : "OFF";

	}

	public void ChooseClass(GameObject activeClass){

		colorClass.SetActive(false);
		wheelClass.SetActive(false);
		upgradesClass.SetActive(false);

		activeClass.SetActive(true);

	}

	public void CheckButtonColors(Button activeButton){

		bodyPaintButton.image.color =
```

```
orgButtonColor;
                rimButton.image.color = orgButtonColor;
                upgradeButton.image.color = orgButtonColor;

                activeButton.image.color = new Color(.65f, 1f,
0f);

        }

        public void ChangeChassisColor (Color color) {

                HR_ModApplier applier =
GameObject.FindObjectOfType<HR_ModApplier>();
                applier.bodyColor = color;
                applier.UpdateStats();

        }

        public void ChangeWheels (int wheelIndex) {

                HR_ModApplier applier =
GameObject.FindObjectOfType<HR_ModApplier>();
                applier.selectedWheel =
HR_Wheels.Instance.wheels[wheelIndex].wheel;
                applier.wheelIndex = wheelIndex;
                applier.UpdateStats();

        }

        public void UpgradeSpeed(){

                HR_ModApplier applier =
GameObject.FindObjectOfType<HR_ModApplier>();
                applier.speedLevel ++;
                applier.UpdateStats();

        }

        public void UpgradeHandling(){

                HR_ModApplier applier =
GameObject.FindObjectOfType<HR_ModApplier>();
                applier.handlingLevel ++;
                applier.UpdateStats();

        }

        public void UpgradeBrake(){

                HR_ModApplier applier =
GameObject.FindObjectOfType<HR_ModApplier>();
                applier.brakeLevel ++;
                applier.UpdateStats();

        }

        public void UpgradeSiren(){

                HR_ModApplier applier =
GameObject.FindObjectOfType<HR_ModApplier>();
                applier.isSirenPurchased = true;
                applier.UpdateStats();
                applier.ToggleSiren();

        }

        public void BuyProperty(int price, string prefsKey){

                int playerCoins =
PlayerPrefs.GetInt("Currency");

                PlayerPrefs.SetInt("Currency", playerCoins -
price);
                PlayerPrefs.SetInt(prefsKey, 1);

        }

}
```

HR_ModificationColor

```
using UnityEngine;
using UnityEngine.UI;
using System;
using System.Collections;
using System.Collections.Generic;

public class HR_ModificationColor : MonoBehaviour {

        public pickedColor _pickedColor;
        public enum pickedColor{Orange, Red, Green, Blue,
Yellow, Black, White, Cyan, Magenta, Pink}
        public int colorPrice;
        public bool unlocked = false;
        private Text priceLabel;
        private Image priceImage;

        void Start(){

                priceLabel =
GetComponentInChildren<Text>();
                priceImage =
priceLabel.GetComponentInParent<Image>();

        }

        public void OnClick () {

                if(!unlocked){
                        BuyColor();
                        return;
                }

                HR_ModHandler handler =
GameObject.FindObjectOfType<HR_ModHandler>();
                Color selectedColor = new Color();

                switch(_pickedColor){

                case pickedColor.Orange:
                        selectedColor = Color.red +
(Color.green / 2f);
                        break;

                case pickedColor.Red:
                        selectedColor = Color.red;
                        break;

                case pickedColor.Green:
                        selectedColor = Color.green;
                        break;

                case pickedColor.Blue:
                        selectedColor = Color.blue;
                        break;

                case pickedColor.Yellow:
                        selectedColor = Color.yellow;
                        break;

                case pickedColor.Black:
                        selectedColor = Color.black;
                        break;
```

```
                case pickedColor.White:
                        selectedColor = Color.white;
                        break;

                case pickedColor.Cyan:
                        selectedColor = Color.cyan;
                        break;

                case pickedColor.Magenta:
                        selectedColor = Color.magenta;
                        break;

                case pickedColor.Pink:
                        selectedColor = new Color(1, 0f,
.5f);
                        break;

                }

                handler.ChangeChassisColor(selectedColor);

        }

        void Update(){

                string currentColorString =
_pickedColor.ToString();

                if(colorPrice <= 0 && !unlocked){

        PlayerPrefs.SetInt(GameObject.FindObjectOfType<RCC
_CarControllerV3>().transform.name + "OwnedColor" +
currentColorString, 1);
                        unlocked = true;
                }


        if(PlayerPrefs.HasKey(GameObject.FindObjectOfType<
RCC_CarControllerV3>().transform.name + "OwnedColor" +
currentColorString))
                        unlocked = true;
                else
                        unlocked = false;

                if(!unlocked){

        if(!priceImage.gameObject.activeSelf)

        priceImage.gameObject.SetActive(true);
                        if(priceLabel.text !=
colorPrice.ToString())
                                priceLabel.text =
colorPrice.ToString();
                }else{

        if(priceImage.gameObject.activeSelf)

        priceImage.gameObject.SetActive(false);
                        if(priceLabel.text !=
"UNLOCKED")
                                priceLabel.text =
"UNLOCKED";
                }

        }

        void BuyColor(){

                int playerCoins =
PlayerPrefs.GetInt("Currency");
                string currentColorString =
_pickedColor.ToString();

                        if(playerCoins >= colorPrice){
                                HR_ModHandler handler =
GameObject.FindObjectOfType<HR_ModHandler>();
                                handler.BuyProperty(colorPrice,
GameObject.FindObjectOfType<RCC_CarControllerV3>().transfor
m.name + "OwnedColor" + currentColorString);
                        }

        }

}
```

HR_RoadPooling

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
//using VacuumShaders.CurvedWorld;
public class HR_RoadPooling : MonoBehaviour {

   public GameObject[] CheckPoint;
   public Camera mainCamera;
        public GameObject wrongWayCollider;
        internal Transform reference;
        private bool animateNow = true;

        [System.Serializable]
        public class RoadObjects{
                public GameObject roadObject;
        }

        public int roadAmountInPool = 10;
        private float[] roadLength;
        [Header("Use This Layer On Road For Calculating Road
Length")] public LayerMask asphaltLayer;

        [Header("Pooling Road Objects. Select Them While They
Are On Your Scene")] public RoadObjects[] roadObjects;
        internal List<GameObject> roads = new
List<GameObject>();

        public float roadWidth = 13.5f;

        public bool automaticRoadLength = true;
        public float manualRoadLength = 60f;
        private int index = 0;
        public GameObject roadLl;
        public GameObject curvedController;
        void Awake () {

      reference = mainCamera.transform;
                roadLength = new float[roadObjects.Length];

                for (int i = 0; i < roadObjects.Length; i++) {
                        //roadLength[i] =
GetRoadLength(i);
                        if(automaticRoadLength)
                                roadLength[i] =
GetRoadLength(i);
                        else
                                roadLength[i] =
manualRoadLength;
                }
                Invoke("CreateRoads",0.2f);
        }
        void Start()
        {
                if (HR_GamePlayHandler.isCheckPoint) {
                        foreach (GameObject _checkpoint
in CheckPoint) {
```

```
                        //              _checkpoint.SetActive
(true);
                        }
                }
                else {
                        foreach (GameObject _checkpoint
in CheckPoint) {
                                //_checkpoint.SetActive
(false);
                        }
                }

        }
        protected float GetRoadLength (int roadIndex){

                GameObject roadReference =
(GameObject)GameObject.Instantiate(roadObjects[roadIndex].roadO
bject, Vector3.zero, Quaternion.identity);

                Bounds combinedBounds =
roadReference.GetComponentInChildren<Renderer>().bounds;
                Renderer[] renderers =
roadReference.GetComponentsInChildren<Renderer>();

                foreach (Renderer render in renderers) {
                        if (render !=
roadReference.GetComponent<Renderer>() && 1 <<
render.gameObject.layer == asphaltLayer)

        combinedBounds.Encapsulate(render.bounds);
                }

                Destroy(roadReference);
                return combinedBounds.size.z;

        }

        void CreateRoads () {

                GameObject allRoads = new GameObject("All
Roads");
                allRoads.transform.position = Vector3.zero;
                allRoads.transform.rotation =
Quaternion.identity;

                for (int i = 0; i < roadAmountInPool; i++) {

                        for (int k = 0; k <
roadObjects.Length; k++) {

                                GameObject go =
(GameObject)GameObject.Instantiate(roadObjects[k].roadObject,
roadObjects[k].roadObject.transform.position,
roadObjects[k].roadObject.transform.rotation);
                                go.isStatic = false;
                                roads.Add(go);

        HR_SetLightmapsManually.Instance.AlignLightmaps(roa
dObjects[k].roadObject, go);

        go.transform.SetParent(allRoads.transform);
                        }

                }

                for (int i = 0; i < roads.Count; i++) {

                        if(i != 0)

        roads[i].transform.position = new Vector3(0f, 0, roads[i -
1].transform.position.z + roadLength[(index <= 0) ?
roadObjects.Length - 1 : index - 1]);
                        index ++;

                        if(index >= roadObjects.Length)
                                index = 0;

                }

                for (int j = 0; j < roadObjects.Length; j++) {


        if(roadObjects[j].roadObject.activeSelf)

        roadObjects[j].roadObject.SetActive(false);

                }

                index = 0;

        }

        void Update(){

                if(animateNow)
                        AnimateRoads();
        }

        void AnimateRoads () {

                for (int i = 0; i < roads.Count; i++) {

                        if(reference.transform.position.z >
(roads[i].transform.position.z + (roadLength[index] * 2f))){

        roads[i].transform.position = new Vector3(0f, 0,
(roads[i].transform.position.z + (roadLength[index] * roads.Count)));
                        }
                        index ++;

                        if(index >= roadObjects.Length)
                                index = 0;

                }

        }
        int currentRoad,i;
        public static bool isChangeCurveRoad;
        Transform referenceNext;
        void assignRows()
        {
                isChangeCurveRoad = false;
        //
        curvedController.GetComponent<CurvedWorld_Controll
er> ().pivotPoint = roads [currentRoad].transform;
                if (currentRoad < roadAmountInPool - 1) {
                        currentRoad++;
                } else if (currentRoad == roadAmountInPool-1
) {
                        currentRoad = 0;
                }

        }

        void OnDrawGizmos(){

                Gizmos.color = new Color(0f, 1f, 0f, .75f);
                if(roadLl)
                Gizmos.DrawCube(new
Vector3(roadLl.transform.position.x,0f,0f), new Vector3(roadWidth
* 3f, 1f, 10f));
                else
                        Gizmos.DrawCube(Vector3.zero,
```

```
new Vector3(roadWidth * 3f, 1f, 10f));
		}

}
```

HR_ScoreBoard

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
using UnityEngine.Advertisements;

public class HR_Scoreboard : MonoBehaviour {

		public GameObject doubleMoneyButton;
		public GameObject doubleScoreLabel;

		[Header("UI Texts On Scoreboard")]
		public Text totalScore;
		public Text subTotalMoney;
		public Text totalMoney;
		//public static bool endSound = true;
		public Text totalDistance;
		public Text totalNearMiss;
		public Text totalOverspeed;
		public Text totalOppositeDirection;
		public Text totalPuzzle;
		public Text totalDistanceMoney;
		public Text totalNearMissMoney;
		public Text totalOverspeedMoney;
		public Text totalOppositeDirectionMoney;
		public Text totalPuzzleMoney;
		public Text comboCount;
		public Text TotalCombo;
		public Text
TotalOppositeDineros,TotalOppositeCountDinero,TotalDineros;

		public int totalDineroScore;

		bool heyzaponce;
		bool hello;
		int puzzleReward;
		int scoretemp;
		int tempGold;
		public static AudioSource DingSound;
   public bool isVideoWatched = false;
   public GameObject AdButton;
   public void Start()
		{
			puzzleReward = 100;
			//endSound = true;
			DingSound =
GameObject.FindGameObjectWithTag
("Menu").GetComponent<AudioSource>();
		}

		public void Update()
		{
			if (PlayerPrefs.GetInt ("SHARE") == 1) {
				totalMoney.text = (tempGold +
tempGold).ToString ();
				PlayerPrefs.SetInt
("Currency",PlayerPrefs.GetInt("Currency") + tempGold +
tempGold);
				tempGold += tempGold;
				PlayerPrefs.SetInt ("SHARE", 0);
			}
			if (isVideoWatched == true)
      {
        isVideoWatched = false;

        totalMoney.text = (Mathf.Floor(tempGold +
tempGold)).ToString ();
        PlayerPrefs.SetInt
("Currency",PlayerPrefs.GetInt("Currency") + tempGold );
				tempGold += tempGold;
			}
		}


		public int totalDistanceMoneyMP
		{
			get
			{
				return
HR_HighwayRacerProperties.Instance._totalDistanceMoneyMP;
			}
		}
		public int totalNearMissMoneyMP{
			get
			{
				return
HR_HighwayRacerProperties.Instance._totalNearMissMoneyMP;
			}
		}
		public int totalOverspeedMoneyMP{
			get
			{
				return
HR_HighwayRacerProperties.Instance._totalOverspeedMoneyMP;
			}
		}
		public int totalOppositeDirectionMP{
			get
			{
				return
HR_HighwayRacerProperties.Instance._totalOppositeDirectionMP;
			}
		}
		int puzzleCharacters;
		public void DisplayResults(){

			DingSound.Play ();
			HR_PlayerHandler playerHandler =
GameObject.FindObjectOfType<HR_PlayerHandler>();
      if (playerHandler != null)
      {
        totalScore.text =
Mathf.Floor(playerHandler.score).ToString("F0");
        totalDistance.text = (playerHandler.distance).ToString("F2");
        totalNearMiss.text =
(playerHandler.nearMisses).ToString("F0");
        totalOverspeed.text =
(playerHandler.highSpeedTotal).ToString("F1");
        totalOppositeDirection.text =
(playerHandler.opposideDirectionTotal).ToString("F1");
      }
			if (LevelChallange.instance) {
        if (LevelChallange.DailyEvent > 0)
				puzzleCharacters =
LevelChallange.instance.totalPuzzleCharacters;// meChange
      } else {
				puzzleCharacters = 0;
			}
			totalPuzzle.text = puzzleCharacters.ToString
();
      totalDistanceMoney.text = Mathf.Floor(playerHandler.distance
* totalDistanceMoneyMP).ToString("F0");
			totalNearMissMoney.text =
```

```
Mathf.Floor(playerHandler.nearMisses *
totalNearMissMoneyMP).ToString("F0");
                        totalOverspeedMoney.text =
Mathf.Floor(playerHandler.highSpeedTotal *
totalOverspeedMoneyMP).ToString("F0");
                        totalOppositeDirectionMoney.text =
Mathf.Floor(playerHandler.opposideDirectionTotal *
totalOppositeDirectionMP).ToString("F0");
                        if (LevelChallange.instance) {
          totalPuzzleMoney.text = (puzzleCharacters *
puzzleReward).ToString ();
                        }
                        else {
                                totalPuzzleMoney.text =
totalPuzzle.text = puzzleCharacters.ToString ();
        }

                        tempGold = Mathf.FloorToInt((Mathf.Floor
(playerHandler.distance * totalDistanceMoneyMP) +
(playerHandler.nearMisses * totalNearMissMoneyMP) +
Mathf.Floor (playerHandler.highSpeedTotal *
totalOverspeedMoneyMP) + Mathf.Floor
(playerHandler.opposideDirectionTotal *
totalOppositeDirectionMP))+(puzzleCharacters*puzzleReward));
                        totalMoney.text =
(Mathf.Floor(playerHandler.distance * totalDistanceMoneyMP) +
(playerHandler.nearMisses * totalNearMissMoneyMP) +
Mathf.Floor(playerHandler.highSpeedTotal *
totalOverspeedMoneyMP) +
Mathf.Floor(playerHandler.opposideDirectionTotal *
totalOppositeDirectionMP)+(puzzleCharacters*puzzleReward)).ToSt
ring("F0");
                        PlayerPrefs.SetInt("Currency",
PlayerPrefs.GetInt("Currency", 0) +
Mathf.FloorToInt(Mathf.Floor(playerHandler.distance *
totalDistanceMoneyMP) + (playerHandler.nearMisses *
totalNearMissMoneyMP) +
Mathf.Floor(playerHandler.highSpeedTotal *
totalOverspeedMoneyMP)+(puzzleCharacters*puzzleReward)));

                        scoretemp =
Mathf.FloorToInt(Mathf.Floor(playerHandler.distance *
totalDistanceMoneyMP) + (playerHandler.nearMisses *
totalNearMissMoneyMP) +
Mathf.Floor(playerHandler.highSpeedTotal *
totalOverspeedMoneyMP)+(puzzleCharacters*puzzleReward));

                        comboCount.text =
HR_PlayerHandler.Instance.maxCombo.ToString("F0");
                        TotalCombo.text =
HR_PlayerHandler.Instance.maxCombo.ToString("F0");

                        TotalOppositeDineros.text =
HR_PlayerHandler.Instance.wrongWayScore.ToString("F0");
                        TotalOppositeCountDinero.text =
HR_PlayerHandler.Instance.wrongWayScore.ToString("F0");

                        totalDineroScore =
HR_PlayerHandler.Instance.maxCombo +
HR_PlayerHandler.Instance.wrongWayScore;

                        TotalDineros.text =
totalDineroScore.ToString("F0");

            PlayerPrefs.SetInt("combo",PlayerPrefs.GetInt("combo")
+totalDineroScore);
                        //              comboCount.text =
PlayerPrefs.GetInt("combo").ToString("F0");

                        if (PlayerPrefs.GetInt ("DOUBLESCORE") ==
1) {
                                PlayerPrefs.SetInt("Currency",
PlayerPrefs.GetInt("Currency", 0) +
Mathf.FloorToInt(Mathf.Floor(playerHandler.distance *
totalDistanceMoneyMP) + (playerHandler.nearMisses *
totalNearMissMoneyMP) +
Mathf.Floor(playerHandler.highSpeedTotal *
totalOverspeedMoneyMP)));
                                        doubleMoneyButton.SetActive
(false);
                        }

                        PlayerPrefs.SetInt ("TEMPSCORE",
scoretemp);

                        gameObject.BroadcastMessage("Animate");

            gameObject.BroadcastMessage("GetNumber");

                        playerHandler.gameObject.SetActive (false);

            }

            public void ViewVideoAd()
            {
//#if !UNITY_EDITOR

      // ShowOptions options = new ShowOptions();
        //  options.resultCallback = HandleShowResult;

          //if (Advertisement.IsReady("rewardedVideo"))
          //{
          //    Advertisement.Show("rewardedVideo", options);
          //}
      }

//      private void HandleShowResult(ShowResult result)
//      {
//          switch (result)
//          {
//              case ShowResult.Finished:
//                  isVideoWatched = true;
//
            AdButton.GetComponent<Button>().interactable=false;
//                  break;
//              case ShowResult.Skipped:
//                  isVideoWatched = true;
//
            AdButton.GetComponent<Button>().interactable=false;
//                  break;
//              case ShowResult.Failed:
//                  break;
//          }
////#endif
//    }


            public void ShowAD(string zone =
"rewardedVideoZone")
            {
                        //ShowOptions options = new ShowOptions();
                        //options.resultCallback = AdCallbackhanler;
                        //Advertisement.Show (zone, options);
            }

            void endAudio()
            {
                        DingSound.Stop ();
            }

}
```

HR_TrafficCar

```
using UnityEngine;
using System.Collections;
//using VacuumShaders.CurvedWorld;

[RequireComponent(typeof(Rigidbody))]
public class HR_TrafficCar : MonoBehaviour
{

    private Rigidbody rigid;
    public bool changingLanes = false;
    public bool immobilized = false;
    public BoxCollider bodyCollider;
    internal BoxCollider triggerCollider;

    public ChangingLines changingLines;
    public enum ChangingLines { Straight, Right, Left }
    internal int currentLine = 0;

    public float maximumSpeed = 10f;
    private float _maximumSpeed = 10f;
    private float desiredSpeed;
    public float distance = 0f;
    private Quaternion steeringAngle = Quaternion.identity;

    public Transform[] wheelModels;
    private float wheelRotation = 0f;

    [Header("Just Lights. All of them will work on "NOT Important" Render Mode.")]
    public LensFlare[] headLights;
    public Light[] brakeLights;
    public LensFlare[] signalLights;
    private bool headlightsOn = false;
    private bool brakingOn = false;

    private SignalsOn signalsOn;
    private enum SignalsOn { Off, Right, Left, All }
    private float signalTimer = 0f;
    private float spawnProtection = 0f;
    public bool allowNearMiss = true;
    public bool isAnimating;
    [Space(10)]

    public AudioClip engineSound;
    private AudioSource engineSoundSource;

    void Awake()
    {

        rigid = GetComponent<Rigidbody>();
        rigid.drag = 1f;
        rigid.angularDrag = 4f;
        rigid.maxAngularVelocity = 2.5f;

        Light[] allLights = GetComponentsInChildren<Light>();

        foreach (Light l in allLights)
        {
            l.renderMode = LightRenderMode.ForceVertex;
            l.cullingMask = 0;
        }

        distance = 50f;

        if (!bodyCollider)
        {
            //Debug.LogWarning (transform.name + "is missing collider in HR_TrafficCar script. Select your vehicle collider. Assigning collider automatically now, but it may select wrong collider...");
            bodyCollider = GetComponentInChildren<BoxCollider>();
        }

        GameObject triggerColliderGO = new GameObject("TriggerVolume");
        triggerColliderGO.transform.position = bodyCollider.bounds.center;
        triggerColliderGO.transform.rotation = bodyCollider.transform.rotation;
        triggerColliderGO.transform.SetParent(transform, true);
        triggerColliderGO.transform.localScale = bodyCollider.transform.localScale;
        triggerColliderGO.AddComponent<BoxCollider>();
        triggerColliderGO.GetComponent<BoxCollider>().isTrigger = true;
        triggerColliderGO.GetComponent<BoxCollider>().size = bodyCollider.size;
        triggerColliderGO.GetComponent<BoxCollider>().center = bodyCollider.center;

        triggerCollider = triggerColliderGO.GetComponent<BoxCollider>();
        triggerCollider.size = new Vector3(bodyCollider.size.x * 1.5f, bodyCollider.size.y, bodyCollider.size.z + (bodyCollider.size.z * 3f));
        triggerCollider.center = new Vector3(bodyCollider.center.x, 0f, bodyCollider.center.z + (triggerCollider.size.z / 2f) - (bodyCollider.size.z / 2f));
        if (HR_GamePlayHandler.Instance.dayOrNight == HR_GamePlayHandler.DayOrNight.Night || HR_GamePlayHandler.Instance.dayOrNight == HR_GamePlayHandler.DayOrNight.Rainy)
            headlightsOn = true;
        else
            headlightsOn = false;

        engineSoundSource = RCC_CreateAudioSource.NewAudioSource(gameObject, "Engine Sound", 2f, 5f, 1f, engineSound, true, true, false);
        engineSoundSource.transform.localPosition = new Vector3(engineSoundSource.transform.localPosition.x, GameObject.FindObjectOfType<SmoothFollow>().topHeight, -GameObject.FindObjectOfType<SmoothFollow>().topDistance);
        engineSoundSource.pitch = 1.5f;

        rigid.constraints = RigidbodyConstraints.FreezeRotationX | RigidbodyConstraints.FreezeRotationZ | RigidbodyConstraints.FreezePositionY;

        _maximumSpeed = maximumSpeed;

        Transform[] allTransforms = GetComponentsInChildren<Transform>();

        //foreach (Transform t in allTransforms) {
        //      t.gameObject.layer = (int)Mathf.Log(HR_HighwayRacerProperties.Instance.trafficCarsLayer.value, 2);
        //}

        //                  triggerCollider.gameObject.layer = LayerMask.NameToLayer("TrafficCarVolume");
        triggerCollider.gameObject.layer = LayerMask.NameToLayer("TransparentFX");

    }
    void Start()
    {
        //                  for (int i = 0; i < headLights.Length; i++) {
        //
        //                              headLights [i].renderMode = LightRenderMode.ForceVertex;
        //
        //                  }
```

```
        for (int i = 0; i < brakeLights.Length; i++)
        {

            brakeLights[i].renderMode = LightRenderMode.ForceVertex;

        }

        //                  for (int i = 0; i < signalLights.Length; i++) {

        //                          signalLights[i].renderMode =
LightRenderMode.ForceVertex;
        //                  }

    }

    void Update()
    {
        if (CarController.canControl)
        {
            spawnProtection += Time.deltaTime;

            Lights();
            Wheels();
        }
        if (General.levelComplete || General.gameOver)
        {
            engineSoundSource.volume = 0f;
        }
    }

    void Lights()
    {

        signalTimer += Time.deltaTime;

        for (int i = 0; i < signalLights.Length; i++)
        {

            if (signalsOn == SignalsOn.Off)
            {
                signalLights[i].brightness = 0f;
            }

            if (signalsOn == SignalsOn.Left)
            {
                if (signalTimer >= .5f)
                {
                    if (signalLights[i].transform.localPosition.x < 0f)
                    {
                        signalLights[i].brightness = 0f;
                    }
                }
                else
                {
                    if (signalLights[i].transform.localPosition.x < 0f)
                    {
                        signalLights[i].brightness = 0.5f; //1f
                    }
                }
                if (signalTimer >= 1f)
                    signalTimer = 0f;
            }

            if (signalsOn == SignalsOn.Right)
            {
                if (signalTimer >= .5f)
                {
                    if (signalLights[i].transform.localPosition.x > 0f)
                    {
                        signalLights[i].brightness = 0f;
                    }
                }
                else
                {
                    if (signalLights[i].transform.localPosition.x > 0f)
                    {
                        signalLights[i].brightness = 0.5f; //1f
                    }
                }
                if (signalTimer >= 1f)
                    signalTimer = 0f;
            }

            if (signalsOn == SignalsOn.All)
            {
                if (signalTimer >= .5f)
                {
                    signalLights[i].brightness = 0f;
                }
                else
                {
                    signalLights[i].brightness = 0.5f; //1f
                }
                if (signalTimer >= 1f)
                    signalTimer = 0f;
            }

        }

        for (int i = 0; i < headLights.Length; i++)
        {

            if (!headlightsOn)
                headLights[i].brightness =
Mathf.Lerp(headLights[i].brightness, 0f, Time.deltaTime * 10f);
            else
                headLights[i].brightness =
Mathf.Lerp(headLights[i].brightness, 0.5f, Time.deltaTime * 10f);

        }

        for (int i = 0; i < brakeLights.Length; i++)
        {

            if (brakingOn)
            {
                brakeLights[i].intensity =
Mathf.Lerp(brakeLights[i].intensity, 1f, Time.deltaTime * 10f);
            }
            else
            {
                if (!headlightsOn)
                    brakeLights[i].intensity =
Mathf.Lerp(brakeLights[i].intensity, 0f, Time.deltaTime * 10f);
                else
                    brakeLights[i].intensity =
Mathf.Lerp(brakeLights[i].intensity, .3f, Time.deltaTime * 10f);
            }

        }

    }

    void Wheels()
    {

        for (int i = 0; i < wheelModels.Length; i++)
        {

            wheelRotation += desiredSpeed * 20 * Time.deltaTime;
            wheelModels[i].transform.localRotation =
Quaternion.Euler(wheelRotation, 0f, 0f);
```

```
    }

  }

  void FixedUpdate()
  {


    if (CarController.canControl)
    {
      if (!immobilized)
      {
        desiredSpeed = Mathf.Clamp(maximumSpeed -
Mathf.Lerp(maximumSpeed, 0f, (distance - 10f) / 50f), 0f,
maximumSpeed);
      }
      else
      {
        desiredSpeed = Mathf.Lerp(desiredSpeed, 0f,
Time.fixedDeltaTime);
      }

      if (distance < 50)
        brakingOn = true;
      else
        brakingOn = false;
      distance = Mathf.Lerp(distance, 50, 0.2f);

      if (!immobilized && HR_GamePlayHandler.Instance.mode
!= HR_GamePlayHandler.Mode.TwoWay)
        transform.rotation = Quaternion.Lerp(transform.rotation,
steeringAngle, Time.fixedDeltaTime * 3f);

      rigid.velocity = Vector3.Lerp(rigid.velocity,
transform.forward * desiredSpeed, Time.fixedDeltaTime * 3f);
      rigid.angularVelocity = Vector3.Lerp(rigid.angularVelocity,
Vector3.zero, Time.fixedDeltaTime * 10f);

      if (!immobilized && HR_GamePlayHandler.Instance.mode
!= HR_GamePlayHandler.Mode.TwoWay)
      {

        switch (changingLines)
        {

          case ChangingLines.Straight:
            steeringAngle = Quaternion.identity;
            transform.position = new
Vector3(Mathf.Lerp(transform.position.x,
HR_TrafficPooling.Instance.lines[currentLine].position.x, 0.1f),
transform.position.y, transform.position.z);
            rigid.velocity = Vector3.Lerp(rigid.velocity,
transform.forward * maximumSpeed, Time.fixedDeltaTime * 3f);
            break;

          case ChangingLines.Left:

            if (currentLine == 0)
            {
              changingLines = ChangingLines.Straight;
              break;
            }

            if (transform.position.x <=
HR_TrafficPooling.Instance.lines[currentLine - 1].position.x + .6f)
            {
              currentLine--;
              signalsOn = SignalsOn.Off;
              changingLines = ChangingLines.Straight;
              steeringAngle = Quaternion.identity;
              //SetSpeed();
            }
            else
            {
              steeringAngle = Quaternion.identity *
Quaternion.Euler(0f, -5f, 0f);
              signalsOn = SignalsOn.Left;
            }
            break;

          case ChangingLines.Right:

            if (currentLine ==
(HR_TrafficPooling.Instance.lines.Length - 1))
            {
              changingLines = ChangingLines.Straight;
              break;
            }

            if (transform.position.x >=
HR_TrafficPooling.Instance.lines[currentLine + 1].position.x - .5f)
            {
              currentLine++;
              signalsOn = SignalsOn.Off;
              changingLines = ChangingLines.Straight;
              steeringAngle = Quaternion.identity;
              //SetSpeed();
            }
            else
            {
              steeringAngle = Quaternion.identity *
Quaternion.Euler(0f, 5f, 0f);
              signalsOn = SignalsOn.Right;
            }
            break;

        }

      }
    }


  }
  void OnTriggerStay(Collider col)
  {

    if ((1 << col.gameObject.layer) !=
HR_HighwayRacerProperties.Instance.trafficCarsLayer.value ||
col.isTrigger)
      return;

    distance = Vector3.Distance(transform.position,
col.transform.position);

  }

  void OnTriggerExit(Collider col)
  {

    if ((1 << col.gameObject.layer) !=
HR_HighwayRacerProperties.Instance.trafficCarsLayer.value)
      return;

  }

  void OnCollisionEnter(Collision col)
  {
    if (immobilized || spawnProtection < .5f || col.collider.tag !=
"Player")
      return;

    immobilized = true;
    signalsOn = SignalsOn.All;
    InvokeRepeating("immobiliz", 1f, 5f);
```

```
    }
    void immobiliz()
    {
        immobilized = false;
    }

    public void OnReAligned()
    {
        //SetSpeed();
        immobilized = false;
        spawnProtection = 0f;
        rigid.velocity = transform.forward * maximumSpeed;
        rigid.angularVelocity = Vector3.zero;
        signalsOn = SignalsOn.Off;
        changingLines = ChangingLines.Straight;
        distance = 50f;

    }

    void SpeedUp()
    {

        distance = 50f;

    }

    public void ChangeLines()
    {

        if (changingLines == ChangingLines.Left || changingLines ==
ChangingLines.Right)
            return;

        int randomNumber = Random.Range(0, 2);

        changingLines = randomNumber == 0 ? ChangingLines.Left :
ChangingLines.Right;

    }
    void SetSpeed()
    {
        if (MenuScript.selectedCar == 0 && MenuScript.selectedCar
<= 2)
        {
            if (currentLine == 0 || currentLine == 2)
            {
                maximumSpeed = 40;
                //                                  Debug.Log ("fast lane");
            }
            else
            {
                maximumSpeed = 30f;
                //                                  Debug.Log ("slow
lane");
            }
        }
        if (MenuScript.selectedCar >= 3 && MenuScript.selectedCar
<= 5)
        {
            if (currentLine == 0 || currentLine == 2)
            {
                maximumSpeed = 45;
                //              Debug.Log("fast lane");
            }
            else
            {
                maximumSpeed = 35f;
                //              Debug.Log("slow lane");
            }
        }
        if (MenuScript.selectedCar >= 6)
        {
            if (currentLine == 0 || currentLine == 2)
            {
                maximumSpeed = 50;
                //              Debug.Log("fast lane");
            }
            else
            {
                maximumSpeed = 40f;
                //              Debug.Log("slow lane");
            }
        }
        //else if (transform.position.x < -1.7f && transform.position.x >
-5.2f) {
        //                              maximumSpeed = 37.5f;
        //                              Debug.Log ("2th lane slow");
        //                  } else {
        //                              maximumSpeed = 51;
        //                              Debug.Log ("1st lane fast");
        //                  }
        _maximumSpeed = maximumSpeed;
        //maximumSpeed = Random.Range(_maximumSpeed,
_maximumSpeed * 1.5f);
    }
    public void ChangeLanes()
    {
        changingLanes = true;
        InvokeRepeating("ChangeLines", 10, 25);
    }
}
```

HR_TrafficPooling

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

public class HR_TrafficPooling : MonoBehaviour
{

    public Camera mainCamera;

    #region SINGLETON PATTERN
    public static HR_TrafficPooling instance;
    public static HR_TrafficPooling Instance
    {
        get
        {
            if (instance == null)
                instance =
GameObject.FindObjectOfType<HR_TrafficPooling>();
            return instance;
        }
    }
    #endregion

    internal Transform reference;
    private Transform _player;
    public Transform[] lines;
    [Range(0, 10)] public int density;
    private bool animateNow
    {
        get
        {
            return CarController.canControl;
        }
    }

    public TrafficCars[] trafficCars;

    [System.Serializable]
    public class TrafficCars
    {
```

```
        public GameObject trafficCar;
        public int frequence = 1;
    }

    public List<HR_TrafficCar> _trafficCars = new
List<HR_TrafficCar>();
    public static HR_TrafficPooling Inctance;
    void Start()
    {
        Inctance = this;
        reference = mainCamera.transform;
        _player = HR_GamePlayHandler.Instance.player.transform;
        CreateTraffic();
        AnimateTraffic();

    }

    void Update()
    {
        if (animateNow)
        {
            AnimateTraffic();
        }

    }
    void CreateTraffic()
    {


        for (int i = 0; i < density; i++)
        { //trafficCars.Length(replaced)

            for (int k = 0; k < trafficCars[i].frequence; k++)
            {
                GameObject go =
(GameObject)GameObject.Instantiate(trafficCars[i].trafficCar,
trafficCars[i].trafficCar.transform.position,
                trafficCars[i].trafficCar.transform.rotation);
                _trafficCars.Add(go.GetComponent<HR_TrafficCar>());
                go.SetActive(false);
                // only level two // List Index 7,14
                if (LevelChallange._level == 11)
                {
                    if (go.transform.GetChild(1).gameObject.tag ==
"Destroycar")
                    {

go.transform.GetChild(1).gameObject.SetActive(true);
                    }
                }
            }

        }

    }

    void AnimateTraffic()
    {
        for (int i = 0; i < _trafficCars.Count; i++)
        {
            if (_trafficCars[i].isAnimating == false)
            {
                if (_player.transform.position.z >
(_trafficCars[i].transform.position.z + 15) ||
                _player.transform.position.z <
(_trafficCars[i].transform.position.z - (325)))
                {
                    ReAlignTraffic(_trafficCars[i]);
                }
            }

        }
    }

    void ReAlignTraffic(HR_TrafficCar realignableObject)
    {

        if (!realignableObject.gameObject.activeSelf)
            realignableObject.gameObject.SetActive(true);
        realignableObject.GetComponent<HR_TrafficCar>().enabled =
true;

realignableObject.GetComponent<HR_TrafficCar>().immobilized =
false;
        realignableObject.GetComponent<Rigidbody>().isKinematic =
false;

realignableObject.transform.GetChild(0).gameObject.SetActive(false
);

        int randomLine = Random.Range(0, lines.Length);
        realignableObject.currentLine = randomLine;
        float randomPositionZ = Random.Range(250, 400);

        realignableObject.transform.position = new
Vector3(lines[randomLine].position.x,
realignableObject.transform.position.y,
        (reference.transform.position.z + randomPositionZ));

        realignableObject.allowNearMiss = true;

        switch (HR_GamePlayHandler.Instance.mode)
        {

            case (HR_GamePlayHandler.Mode.OneWay):
                realignableObject.transform.rotation = Quaternion.identity;
                break;
            case (HR_GamePlayHandler.Mode.TwoWay):
                if (realignableObject.transform.position.x <= 0f)
                    realignableObject.transform.rotation =
Quaternion.identity * Quaternion.Euler(0f, 180f, 0f);
                else
                    realignableObject.transform.rotation =
Quaternion.identity;
                break;
            case (HR_GamePlayHandler.Mode.TimeAttack):
                realignableObject.transform.rotation = Quaternion.identity;
                break;
            case (HR_GamePlayHandler.Mode.Bomb):
                realignableObject.transform.rotation = Quaternion.identity;
                break;

        }

        realignableObject.OnReAligned();

        if (CheckIfClipping(realignableObject.triggerCollider))
        {
            realignableObject.gameObject.SetActive(false);
        }
    }


    bool CheckIfClipping(BoxCollider trafficCarBound)
    {

        for (int i = 0; i < _trafficCars.Count; i++)
        {
            if (_trafficCars[i].isAnimating == false)
            {
                if
(!trafficCarBound.transform.IsChildOf(_trafficCars[i].transform)
&& _trafficCars[i].gameObject.activeSelf)
```

```
            {
                if
(HR_BoundsExtension.ContainBounds(trafficCarBound.transform,
                trafficCarBound.bounds,
_trafficCars[i].triggerCollider.bounds))
                {
                    return true;
                }
            }
        }
    }

    return false;

  }

}
```

PlayerPrefsX

```
// ArrayPrefs2 v 1.4

using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;

public class PlayerPrefsX
{
        static private int endianDiff1;
        static private int endianDiff2;
        static private int idx;
        static private byte [] byteBlock;

        enum ArrayType {Float, Int32, Bool, String, Vector2,
Vector3, Quaternion, Color}

        public static bool SetBool ( String name, bool value)
        {
                try
                {
                        PlayerPrefs.SetInt(name, value? 1 :
0);
                }
                catch
                {
                        return false;
                }
                return true;
        }

        public static bool GetBool (String name)
        {
                return PlayerPrefs.GetInt(name) == 1;
        }

        public static bool GetBool (String name, bool
defaultValue)
        {
                return (1==PlayerPrefs.GetInt(name,
defaultValue?1:0));
        }

        public static long GetLong(string key, long defaultValue)
        {
                int lowBits, highBits;
                SplitLong(defaultValue, out lowBits, out
highBits);
                lowBits = PlayerPrefs.GetInt(key+"_lowBits",
lowBits);
                highBits =
PlayerPrefs.GetInt(key+"_highBits", highBits);

                // unsigned, to prevent loss of sign bit.
                ulong ret = (uint)highBits;
                ret = (ret << 32);
                return (long)(ret | (ulong)(uint)lowBits);
        }

        public static long GetLong(string key)
        {
                int lowBits =
PlayerPrefs.GetInt(key+"_lowBits");
                int highBits =
PlayerPrefs.GetInt(key+"_highBits");

                // unsigned, to prevent loss of sign bit.
                ulong ret = (uint)highBits;
                ret = (ret << 32);
                return (long)(ret | (ulong)(uint)lowBits);
        }

        private static void SplitLong(long input, out int lowBits,
out int highBits)
        {
                // unsigned everything, to prevent loss of sign
bit.
                lowBits = (int)(uint)(ulong)input;
                highBits = (int)(uint)(input >> 32);
        }

        public static void SetLong(string key, long value)
        {
                int lowBits, highBits;
                SplitLong(value, out lowBits, out highBits);
                PlayerPrefs.SetInt(key+"_lowBits", lowBits);
                PlayerPrefs.SetInt(key+"_highBits", highBits);
        }

        public static bool SetVector2 (String key, Vector2 vector)
        {
                return SetFloatArray(key, new float[]{vector.x,
vector.y});
        }

        static Vector2 GetVector2 (String key)
        {
                var floatArray = GetFloatArray(key);
                if (floatArray.Length < 2)
                {
                        return Vector2.zero;
                }
                return new Vector2(floatArray[0],
floatArray[1]);
        }

        public static Vector2 GetVector2 (String key, Vector2
defaultValue)
        {
                if (PlayerPrefs.HasKey(key))
                {
                        return GetVector2(key);
                }
                return defaultValue;
        }

        public static bool SetVector3 (String key, Vector3 vector)
        {
                return SetFloatArray(key, new float
```

```
[]{vector.x, vector.y, vector.z});
	}

	public static Vector3 GetVector3 (String key)
	{
		var floatArray = GetFloatArray(key);
		if (floatArray.Length < 3)
		{
			return Vector3.zero;
		}
		return new Vector3(floatArray[0],
floatArray[1], floatArray[2]);
	}

	public static Vector3 GetVector3 (String key, Vector3
defaultValue)
	{
		if (PlayerPrefs.HasKey(key))
		{
			return GetVector3(key);
		}
		return defaultValue;
	}

	public static bool SetQuaternion (String key, Quaternion
vector)
	{
		return SetFloatArray(key, new float[]{vector.x,
vector.y, vector.z, vector.w});
	}

	public static Quaternion GetQuaternion (String key)
	{
		var floatArray = GetFloatArray(key);
		if (floatArray.Length < 4)
		{
			return Quaternion.identity;
		}
		return new Quaternion(floatArray[0],
floatArray[1], floatArray[2], floatArray[3]);
	}

	public static Quaternion GetQuaternion (String key,
Quaternion defaultValue )
	{
		if (PlayerPrefs.HasKey(key))
		{
			return GetQuaternion(key);
		}
		return defaultValue;
	}

	public static bool SetColor (String key, Color color)
	{
		return SetFloatArray(key, new float[]{color.r,
color.g, color.b, color.a});
	}

	public static Color GetColor (String key)
	{
		var floatArray = GetFloatArray(key);
		if (floatArray.Length < 4)
		{
			return new Color(0.0f, 0.0f, 0.0f,
0.0f);
		}
		return new Color(floatArray[0], floatArray[1],
floatArray[2], floatArray[3]);
	}

	public static Color GetColor (String key , Color
defaultValue )
	{
		if (PlayerPrefs.HasKey(key))
		{
			return GetColor(key);
		}
		return defaultValue;
	}

	public static bool SetBoolArray (String key, bool[]
boolArray)
	{
		// Make a byte array that's a multiple of 8 in
length, plus 5 bytes to store the number of entries as an int32 (+
identifier)
		// We have to store the number of entries, since
the boolArray length might not be a multiple of 8, so there could be
some padded zeroes
		var bytes = new byte[(boolArray.Length + 7)/8
+ 5];
		bytes[0] = System.Convert.ToByte
(ArrayType.Bool);	// Identifier
		var bits = new BitArray(boolArray);
		bits.CopyTo (bytes, 5);
		Initialize();
		ConvertInt32ToBytes (boolArray.Length,
bytes); // The number of entries in the boolArray goes in the first 4
bytes

		return SaveBytes (key, bytes);
	}

	public static bool[] GetBoolArray (String key)
	{
		if (PlayerPrefs.HasKey(key))
		{
			var bytes =
System.Convert.FromBase64String (PlayerPrefs.GetString(key));
			if (bytes.Length < 5)
			{
				Debug.LogError
("Corrupt preference file for " + key);
				return new bool[0];
			}
			if ((ArrayType)bytes[0] !=
ArrayType.Bool)
			{
				Debug.LogError (key + "
is not a boolean array");
				return new bool[0];
			}
			Initialize();

			// Make a new bytes array that
doesn't include the number of entries + identifier (first 5 bytes) and
turn that into a BitArray
			var bytes2 = new
byte[bytes.Length-5];
			System.Array.Copy(bytes, 5,
bytes2, 0, bytes2.Length);
			var bits = new BitArray(bytes2);
			// Get the number of entries from
the first 4 bytes after the identifier and resize the BitArray to that
length, then convert it to a boolean array
			bits.Length = ConvertBytesToInt32
(bytes);
			var boolArray = new
bool[bits.Count];
			bits.CopyTo (boolArray, 0);

			return boolArray;
		}
		return new bool[0];
```

```
	}

	public static bool[] GetBoolArray (String key, bool
defaultValue, int defaultSize)
	{
		if (PlayerPrefs.HasKey(key))
		{
			return GetBoolArray(key);
		}
		var boolArray = new bool[defaultSize];
		for(int i=0;i<defaultSize;i++)
		{
			boolArray[i] = defaultValue;
		}
		return boolArray;
	}

	public static bool SetStringArray (String key, String[]
stringArray)
	{
		var bytes = new byte[stringArray.Length + 1];
		bytes[0] = System.Convert.ToByte
(ArrayType.String);	// Identifier
		Initialize();

		// Store the length of each string that's in
stringArray, so we can extract the correct strings in GetStringArray
		for (var i = 0; i < stringArray.Length; i++)
		{
			if (stringArray[i] == null)
			{
				Debug.LogError ("Can't
save null entries in the string array when setting " + key);
				return false;
			}
			if (stringArray[i].Length > 255)
			{
				Debug.LogError
("Strings cannot be longer than 255 characters when setting " + key);
				return false;
			}
			bytes[idx++] =
(byte)stringArray[i].Length;
		}

		try
		{
			PlayerPrefs.SetString (key,
System.Convert.ToBase64String (bytes) + "|" + String.Join("",
stringArray));
		}
		catch
		{
			return false;
		}
		return true;
	}

	public static String[] GetStringArray (String key)
	{
		if (PlayerPrefs.HasKey(key)) {
			var completeString =
PlayerPrefs.GetString(key);
			var separatorIndex =
completeString.IndexOf("|"[0]);
			if (separatorIndex < 4) {
				Debug.LogError
("Corrupt preference file for " + key);
				return new String[0];
			}
			var bytes =
System.Convert.FromBase64String (completeString.Substring(0,
separatorIndex));
			if ((ArrayType)bytes[0] !=
ArrayType.String) {
				Debug.LogError (key + "
is not a string array");
				return new String[0];
			}
			Initialize();

			var numberOfEntries =
bytes.Length-1;
			var stringArray = new
String[numberOfEntries];
			var stringIndex = separatorIndex +
1;
			for (var i = 0; i < numberOfEntries;
i++)
			{
				int stringLength =
bytes[idx++];
				if (stringIndex +
stringLength > completeString.Length)
				{

	Debug.LogError ("Corrupt preference file for " + key);
					return new
String[0];
				}
				stringArray[i] =
completeString.Substring(stringIndex, stringLength);
				stringIndex +=
stringLength;
			}

			return stringArray;
		}
		return new String[0];
	}

	public static String[] GetStringArray (String key, String
defaultValue, int defaultSize)
	{
		if (PlayerPrefs.HasKey(key))
		{
			return GetStringArray(key);
		}
		var stringArray = new String[defaultSize];
		for(int i=0;i<defaultSize;i++)
		{
			stringArray[i] = defaultValue;
		}
		return stringArray;
	}

	public static bool SetIntArray (String key, int[] intArray)
	{
		return SetValue (key, intArray,
ArrayType.Int32, 1, ConvertFromInt);
	}

	public static bool SetFloatArray (String key, float[]
floatArray)
	{
		return SetValue (key, floatArray,
ArrayType.Float, 1, ConvertFromFloat);
	}

	public static bool SetVector2Array (String key, Vector2[]
vector2Array )
	{
		return SetValue (key, vector2Array,
ArrayType.Vector2, 2, ConvertFromVector2);
```

```
		}

		public static bool SetVector3Array (String key, Vector3[]
vector3Array)
		{
			return SetValue (key, vector3Array,
ArrayType.Vector3, 3, ConvertFromVector3);
		}

		public static bool SetQuaternionArray (String key,
Quaternion[] quaternionArray )
		{
			return SetValue (key, quaternionArray,
ArrayType.Quaternion, 4, ConvertFromQuaternion);
		}

		public static bool SetColorArray (String key, Color[]
colorArray)
		{
			return SetValue (key, colorArray,
ArrayType.Color, 4, ConvertFromColor);
		}

		private static bool SetValue<T> (String key, T array,
ArrayType arrayType, int vectorNumber, Action<T, byte[],int>
convert) where T : IList
		{
			var bytes = new
byte[(4*array.Count)*vectorNumber + 1];
			bytes[0] = System.Convert.ToByte
(arrayType);		// Identifier
			Initialize();

			for (var i = 0; i < array.Count; i++) {
				convert (array, bytes, i);
			}
			return SaveBytes (key, bytes);
		}

		private static void ConvertFromInt (int[] array, byte[]
bytes, int i)
		{
			ConvertInt32ToBytes (array[i], bytes);
		}

		private static void ConvertFromFloat (float[] array, byte[]
bytes, int i)
		{
			ConvertFloatToBytes (array[i], bytes);
		}

		private static void ConvertFromVector2 (Vector2[] array,
byte[] bytes, int i)
		{
			ConvertFloatToBytes (array[i].x, bytes);
			ConvertFloatToBytes (array[i].y, bytes);
		}

		private static void ConvertFromVector3 (Vector3[] array,
byte[] bytes, int i)
		{
			ConvertFloatToBytes (array[i].x, bytes);
			ConvertFloatToBytes (array[i].y, bytes);
			ConvertFloatToBytes (array[i].z, bytes);
		}

		private static void ConvertFromQuaternion (Quaternion[]
array, byte[] bytes, int i)
		{
			ConvertFloatToBytes (array[i].x, bytes);
			ConvertFloatToBytes (array[i].y, bytes);
			ConvertFloatToBytes (array[i].z, bytes);
			ConvertFloatToBytes (array[i].w, bytes);
		}

		private static void ConvertFromColor (Color[] array,
byte[] bytes, int i)
		{
			ConvertFloatToBytes (array[i].r, bytes);
			ConvertFloatToBytes (array[i].g, bytes);
			ConvertFloatToBytes (array[i].b, bytes);
			ConvertFloatToBytes (array[i].a, bytes);
		}

		public static int[] GetIntArray (String key)
		{
			var intList = new List<int>();
			GetValue (key, intList, ArrayType.Int32, 1,
ConvertToInt);
			return intList.ToArray();
		}

		public static int[] GetIntArray (String key, int
defaultValue, int defaultSize)
		{
			if (PlayerPrefs.HasKey(key))
			{
				return GetIntArray(key);
			}
			var intArray = new int[defaultSize];
			for (int i=0; i<defaultSize; i++)
			{
				intArray[i] = defaultValue;
			}
			return intArray;
		}

		public static float[] GetFloatArray (String key)
		{
			var floatList = new List<float>();
			GetValue (key, floatList, ArrayType.Float, 1,
ConvertToFloat);
			return floatList.ToArray();
		}

		public static float[] GetFloatArray (String key, float
defaultValue, int defaultSize)
		{
			if (PlayerPrefs.HasKey(key))
			{
				return GetFloatArray(key);
			}
			var floatArray = new float[defaultSize];
			for (int i=0; i<defaultSize; i++)
			{
				floatArray[i] = defaultValue;
			}
			return floatArray;
		}

		public static Vector2[] GetVector2Array (String key)
		{
			var vector2List = new List<Vector2>();
			GetValue (key, vector2List,
ArrayType.Vector2, 2, ConvertToVector2);
			return vector2List.ToArray();
		}

		public static Vector2[] GetVector2Array (String key,
Vector2 defaultValue, int defaultSize)
		{
			if (PlayerPrefs.HasKey(key))
			{
				return GetVector2Array(key);
```

```
                }
                var vector2Array = new Vector2[defaultSize];
                for(int i=0; i< defaultSize;i++)
                {
                        vector2Array[i] = defaultValue;
                }
                return vector2Array;
        }

        public static Vector3[] GetVector3Array (String key)
        {
                var vector3List = new List<Vector3>();
                GetValue (key, vector3List,
ArrayType.Vector3, 3, ConvertToVector3);
                return vector3List.ToArray();
        }

        public static Vector3[] GetVector3Array (String key,
Vector3 defaultValue, int defaultSize)
        {
                if (PlayerPrefs.HasKey(key))

                {
                        return GetVector3Array(key);
                }
                var vector3Array = new Vector3[defaultSize];
                for (int i=0; i<defaultSize;i++)
                {
                        vector3Array[i] = defaultValue;
                }
                return vector3Array;
        }

        public static Quaternion[] GetQuaternionArray (String
key)
        {
                var quaternionList = new List<Quaternion>();
                GetValue (key, quaternionList,
ArrayType.Quaternion, 4, ConvertToQuaternion);
                return quaternionList.ToArray();
        }

        public static Quaternion[] GetQuaternionArray (String
key, Quaternion defaultValue, int defaultSize)
        {
                if (PlayerPrefs.HasKey(key))
                {
                        return GetQuaternionArray(key);
                }
                var quaternionArray = new
Quaternion[defaultSize];
                for(int i=0;i<defaultSize;i++)
                {
                        quaternionArray[i] = defaultValue;
                }
                return quaternionArray;
        }

        public static Color[] GetColorArray (String key)
        {
                var colorList = new List<Color>();
                GetValue (key, colorList, ArrayType.Color, 4,
ConvertToColor);
                return colorList.ToArray();
        }

        public static Color[] GetColorArray (String key, Color
defaultValue, int defaultSize)
        {
                if (PlayerPrefs.HasKey(key)) {
                        return GetColorArray(key);
                }
                var colorArray = new Color[defaultSize];
                for(int i=0;i<defaultSize;i++)
                {
                        colorArray[i] = defaultValue;
                }
                return colorArray;
        }

        private static void GetValue<T> (String key, T list,
ArrayType arrayType, int vectorNumber, Action<T, byte[]> convert)
where T : IList
        {
                if (PlayerPrefs.HasKey(key))
                {
                        var bytes =
System.Convert.FromBase64String (PlayerPrefs.GetString(key));
                        if ((bytes.Length-1) %
(vectorNumber*4) != 0)
                        {
                                Debug.LogError
("Corrupt preference file for " + key);
                                return;
                        }
                        if ((ArrayType)bytes[0] !=
arrayType)
                        {
                                Debug.LogError (key + "
is not a " + arrayType.ToString() + " array");
                                return;
                        }
                        Initialize();

                        var end = (bytes.Length-1) /
(vectorNumber*4);
                        for (var i = 0; i < end; i++)
                        {
                                convert (list, bytes);
                        }
                }
        }

        private static void ConvertToInt (List<int> list, byte[]
bytes)
        {
                list.Add (ConvertBytesToInt32(bytes));
        }

        private static void ConvertToFloat (List<float> list, byte[]
bytes)
        {
                list.Add (ConvertBytesToFloat(bytes));
        }

        private static void ConvertToVector2 (List<Vector2> list,
byte[] bytes)
        {
                list.Add (new
Vector2(ConvertBytesToFloat(bytes),
ConvertBytesToFloat(bytes)));
        }

        private static void ConvertToVector3 (List<Vector3> list,
byte[] bytes)
        {
                list.Add (new
Vector3(ConvertBytesToFloat(bytes), ConvertBytesToFloat(bytes),
ConvertBytesToFloat(bytes)));
        }

        private static void ConvertToQuaternion
(List<Quaternion> list,byte[] bytes)
        {
```

```
                list.Add (new
Quaternion(ConvertBytesToFloat(bytes),
ConvertBytesToFloat(bytes), ConvertBytesToFloat(bytes),
ConvertBytesToFloat(bytes)));
        }

        private static void ConvertToColor (List<Color> list,
byte[] bytes)
        {
                list.Add (new
Color(ConvertBytesToFloat(bytes), ConvertBytesToFloat(bytes),
ConvertBytesToFloat(bytes), ConvertBytesToFloat(bytes)));
        }

        public static void ShowArrayType (String key)
        {
                var bytes = System.Convert.FromBase64String
(PlayerPrefs.GetString(key));
                if (bytes.Length > 0)
                {
                        ArrayType arrayType =
(ArrayType)bytes[0];
                        Debug.Log (key + " is a " +
arrayType.ToString() + " array");
                }
        }

        private static void Initialize ()
        {
                if (System.BitConverter.IsLittleEndian)
                {
                        endianDiff1 = 0;
                        endianDiff2 = 0;
                }
                else
                {
                        endianDiff1 = 3;
                        endianDiff2 = 1;
                }
                if (byteBlock == null)
                {
                        byteBlock = new byte[4];
                }
                idx = 1;
        }

        private static bool SaveBytes (String key, byte[] bytes)
        {
                try
                {
                        PlayerPrefs.SetString (key,
System.Convert.ToBase64String (bytes));
                }
                catch
                {
                        return false;
                }
                return true;
        }

        private static void ConvertFloatToBytes (float f, byte[]
bytes)
        {
                byteBlock = System.BitConverter.GetBytes
(f);
                ConvertTo4Bytes (bytes);
        }

        private static float ConvertBytesToFloat (byte[] bytes)
        {
                ConvertFrom4Bytes (bytes);
                return System.BitConverter.ToSingle
(byteBlock, 0);
        }

        private static void ConvertInt32ToBytes (int i, byte[]
bytes)
        {
                byteBlock = System.BitConverter.GetBytes
(i);
                ConvertTo4Bytes (bytes);
        }

        private static int ConvertBytesToInt32 (byte[] bytes)
        {
                ConvertFrom4Bytes (bytes);
                return System.BitConverter.ToInt32
(byteBlock, 0);
        }

        private static void ConvertTo4Bytes (byte[] bytes)
        {
                bytes[idx  ] = byteBlock[    endianDiff1];
                bytes[idx+1] = byteBlock[1 + endianDiff2];
                bytes[idx+2] = byteBlock[2 - endianDiff2];
                bytes[idx+3] = byteBlock[3 - endianDiff1];
                idx += 4;
        }

        private static void ConvertFrom4Bytes (byte[] bytes)
        {
                byteBlock[    endianDiff1] = bytes[idx  ];
                byteBlock[1 + endianDiff2] = bytes[idx+1];
                byteBlock[2 - endianDiff2] = bytes[idx+2];
                byteBlock[3 - endianDiff1] = bytes[idx+3];
                idx += 4;
        }
}
```

RCC_Dashboard

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

[AddComponentMenu("BoneCracker Games/Realistic Car
Controller/Misc/Truck Trailer")]
public class RCC_TruckTrailer : MonoBehaviour {

        private RCC_CarControllerV3 carController;
        private Rigidbody rigid;
        public Transform COM;

        //Extra Wheels.
        public WheelCollider[] wheelColliders;
        private List<WheelCollider> leftWheelColliders = new
List<WheelCollider>();
        private List<WheelCollider> rightWheelColliders = new
List<WheelCollider>();

        public float antiRoll = 50000f;

        void Start () {

                rigid = GetComponent<Rigidbody>();
                carController =
transform.GetComponentInParent<RCC_CarControllerV3>();

                GetComponent<Rigidbody>().interpolation =
RigidbodyInterpolation.None;
                GetComponent<Rigidbody>().interpolation =
RigidbodyInterpolation.Interpolate;
                GetComponent<Rigidbody>().centerOfMass =
transform.InverseTransformPoint(COM.transform.position);
```

```
                antiRoll =
carController.antiRollFrontHorizontal;

                for (int i = 0; i < wheelColliders.Length; i++) {


        if(wheelColliders[i].transform.localPosition.x < 0f)

        leftWheelColliders.Add(wheelColliders[i]);
                        else

        rightWheelColliders.Add(wheelColliders[i]);

                }

                gameObject.SetActive (false);
                gameObject.SetActive (true);

        }

        void FixedUpdate(){

                AntiRollBars();

                //Applying Small Torque For Preventing Stuck
Issue. Unity 5 WheelColliders Are Weird :/
                foreach(WheelCollider wc in wheelColliders){
                        wc.motorTorque =
carController._gasInput * (carController.engineTorque / 10f);
                }

        }

        public void AntiRollBars (){

                for (int i = 0; i < leftWheelColliders.Count;
i++) {

                        WheelHit hit;

                        float travelL = 1.0f;
                        float travelR = 1.0f;

                        bool groundedL=
leftWheelColliders[i].GetGroundHit(out hit);

                        if (groundedL)
                                travelL = (-
leftWheelColliders[i].transform.InverseTransformPoint(hit.point).y -
leftWheelColliders[i].radius) /
leftWheelColliders[i].suspensionDistance;

                        bool groundedR=
rightWheelColliders[i].GetGroundHit(out hit);

                        if (groundedR)
                                travelR = (-
rightWheelColliders[i].transform.InverseTransformPoint(hit.point).y
- rightWheelColliders[i].radius) /
rightWheelColliders[i].suspensionDistance;

                        float antiRollForce= (travelL -
travelR) * antiRoll;

                        if (groundedL)

        rigid.AddForceAtPosition(leftWheelColliders[i].transform
.up * -antiRollForce, leftWheelColliders[i].transform.position);
                        if (groundedR)

        rigid.AddForceAtPosition(rightWheelColliders[i].transfor
m.up * antiRollForce, rightWheelColliders[i].transform.position);

                }

        }

}
```

RCC_WheelCollider

```
using UnityEngine;
using System.Linq;
using System.Collections;
using System.Collections.Generic;

[AddComponentMenu("BoneCracker Games/Realistic Car
Controller/Wheel Collider")]
[RequireComponent (typeof(WheelCollider))]
public class RCC_WheelCollider : MonoBehaviour {

        private RCC_CarControllerV3 carController;
        private Rigidbody rigid;

        private WheelCollider _wheelCollider;
        public WheelCollider wheelCollider{
                get{
                        if(_wheelCollider == null)
                                _wheelCollider =
GetComponent<WheelCollider>();
                        return _wheelCollider;
                }set{
                        _wheelCollider = value;
                }
        }

        private List <RCC_WheelCollider> allWheelColliders =
new List<RCC_WheelCollider>() ;
        public Transform wheelModel;

        private float wheelRotation = 0f;
        public float camber = 0f;
        private PhysicMaterial groundMaterial;

        internal bool isGrounded = false;
        internal float totalSlip = 0f;
        internal float rpm = 0f;
        internal float wheelRPMToSpeed = 0f;
        internal float wheelTemparature = 0f;

        private RCC_GroundMaterials
physicsMaterials{get{return RCC_GroundMaterials.Instance;}}
        private RCC_GroundMaterials.GroundMaterialFrictions[]
physicsFrictions{get{return
RCC_GroundMaterials.Instance.frictions;}}

        private RCC_Skidmarks skidmarks;
        private float startSlipValue = .25f;
        private int lastSkidmark = -1;

        private float wheelSlipAmountSideways;
        private float wheelSlipAmountForward;

        private float orgSidewaysStiffness = 1f;
        private float orgForwardStiffness = 1f;

        //WheelFriction Curves and Stiffness.
        public WheelFrictionCurve forwardFrictionCurve;
        public WheelFrictionCurve sidewaysFrictionCurve;

        private AudioSource audioSource;
        private AudioClip audioClip;
```

```
	private List<ParticleSystem> allWheelParticles = new
List<ParticleSystem>();
	private ParticleSystem.EmissionModule emission;

	internal float tractionHelpedSidewaysStiffness = 1f;

	void Awake (){

		carController =
GetComponentInParent<RCC_CarControllerV3>();
		rigid = GetComponentInParent<Rigidbody>();
		wheelCollider =
GetComponent<WheelCollider>();

		if (!RCC_Settings.Instance.dontUseSkidmarks)
{
			if (FindObjectOfType
(typeof(RCC_Skidmarks))) {
				skidmarks =
FindObjectOfType (typeof(RCC_Skidmarks)) as RCC_Skidmarks;
			} else {
				Debug.Log ("No
skidmarks object found. Creating new one...");
				skidmarks =
(RCC_Skidmarks)Instantiate
(RCC_Settings.Instance.skidmarksManager, Vector3.zero,
Quaternion.identity);
			}
		}

		wheelCollider.mass = rigid.mass / 15f;
		forwardFrictionCurve =
wheelCollider.forwardFriction;
		sidewaysFrictionCurve =
wheelCollider.sidewaysFriction;
		camber = this ==
carController.FrontLeftWheelCollider || this ==
carController.FrontRightWheelCollider ? carController.frontCamber
: carController.rearCamber;

		switch(RCC_Settings.Instance.behaviorType){

			case
RCC_Settings.BehaviorType.HighwayRacer:
				forwardFrictionCurve =
SetFrictionCurves(forwardFrictionCurve, .25f, 1f, .8f, 1f);
				sidewaysFrictionCurve =
SetFrictionCurves(sidewaysFrictionCurve, .25f, 1.5f, 1f, 1f);

	wheelCollider.forceAppPointDistance =
Mathf.Clamp(wheelCollider.forceAppPointDistance, .2f, 1f);
				break;

		}

		orgForwardStiffness =
forwardFrictionCurve.stiffness;
		orgSidewaysStiffness =
sidewaysFrictionCurve.stiffness;
		wheelCollider.forwardFriction =
forwardFrictionCurve;
		wheelCollider.sidewaysFriction =
sidewaysFrictionCurve;


	if(RCC_Settings.Instance.useSharedAudioSources){

	if(!carController.transform.Find("All Audio Sources/Skid
Sound AudioSource"))
				audioSource =
RCC_CreateAudioSource.NewAudioSource(carController.gameObje
ct, "Skid Sound AudioSource", 5, 50, 0, audioClip, true, true, false);
			else
				audioSource =
carController.transform.Find("All Audio Sources/Skid Sound
AudioSource").GetComponent<AudioSource>();
		}else{
				audioSource =
RCC_CreateAudioSource.NewAudioSource(carController.gameObje
ct, "Skid Sound AudioSource", 5, 50, 0, audioClip, true, true, false);
				audioSource.transform.position =
transform.position;
		}

		if
(!RCC_Settings.Instance.dontUseAnyParticleEffects) {

			for (int i = 0; i <
RCC_GroundMaterials.Instance.frictions.Length; i++) {

				GameObject ps =
(GameObject)Instantiate (RCC_GroundMaterials.Instance.frictions
[i].groundParticles, transform.position, transform.rotation) as
GameObject;
				emission =
ps.GetComponent<ParticleSystem> ().emission;
				emission.enabled = false;
				ps.transform.SetParent
(transform, false);

	ps.transform.localPosition = Vector3.zero;

	ps.transform.localRotation = Quaternion.identity;
				allWheelParticles.Add
(ps.GetComponent<ParticleSystem> ());

			}

		}

	}

	void Start(){

		allWheelColliders =
carController.allWheelColliders.ToList();
		allWheelColliders.Remove(this);

	}

	WheelFrictionCurve
SetFrictionCurves(WheelFrictionCurve curve, float extremumSlip,
float extremumValue, float asymptoteSlip, float asymptoteValue){

		WheelFrictionCurve newCurve = curve;

		newCurve.extremumSlip = extremumSlip;
		newCurve.extremumValue = extremumValue;
		newCurve.asymptoteSlip = asymptoteSlip;
		newCurve.asymptoteValue = asymptoteValue;

		return newCurve;

	}

	void Update(){

		if (!carController.enabled)
			return;

		if(!carController.sleepingRigid){

			WheelAlign();
			WheelCamber();
```

```
                }

        }

        void  FixedUpdate (){

                if (!carController.enabled)
                        return;

                WheelHit hit;
                isGrounded =
wheelCollider.GetGroundHit(out hit);

                rpm = wheelCollider.rpm;
                wheelRPMToSpeed = (((wheelCollider.rpm *
wheelCollider.radius) / 2.8f) * Mathf.Lerp(1f, .75f, hit.forwardSlip))
* rigid.transform.lossyScale.y;

                SkidMarks();
                Frictions();
                Audio();

        }

        public void WheelAlign (){

                if(!wheelModel){
                        Debug.LogError(transform.name +
" wheel of the " + carController.transform.name + " is missing wheel
model. This wheel is disabled");
                        enabled = false;
                        return;
                }

                RaycastHit hit;
                WheelHit CorrespondingGroundHit;

                Vector3 ColliderCenterPoint =
wheelCollider.transform.TransformPoint(wheelCollider.center);
                wheelCollider.GetGroundHit(out
CorrespondingGroundHit);

                if(Physics.Raycast(ColliderCenterPoint, -
wheelCollider.transform.up, out hit,
(wheelCollider.suspensionDistance + wheelCollider.radius) *
transform.localScale.y) && hit.transform.gameObject.layer !=
(int)Mathf.Log(RCC_Settings.Instance.vehicleLayer.value, 2) &&
!hit.collider.isTrigger){
                        wheelModel.transform.position =
hit.point + (wheelCollider.transform.up * wheelCollider.radius) *
transform.localScale.y;
                        float extension = (-
wheelCollider.transform.InverseTransformPoint(CorrespondingGrou
ndHit.point).y - wheelCollider.radius) /
wheelCollider.suspensionDistance;

        Debug.DrawLine(CorrespondingGroundHit.point,
CorrespondingGroundHit.point + wheelCollider.transform.up *
(CorrespondingGroundHit.force / rigid.mass), extension <= 0.0 ?
Color.magenta : Color.white);

        Debug.DrawLine(CorrespondingGroundHit.point,
CorrespondingGroundHit.point - wheelCollider.transform.forward *
CorrespondingGroundHit.forwardSlip * 2f, Color.green);

        Debug.DrawLine(CorrespondingGroundHit.point,
CorrespondingGroundHit.point - wheelCollider.transform.right *
CorrespondingGroundHit.sidewaysSlip * 2f, Color.red);
                }else{
                        wheelModel.transform.position =
Vector3.Lerp(wheelModel.transform.position, ColliderCenterPoint -
(wheelCollider.transform.up * wheelCollider.suspensionDistance) *
transform.localScale.y, Time.deltaTime * 10f);
                }

                wheelRotation += wheelCollider.rpm * 6 *
Time.deltaTime;
                wheelModel.transform.rotation =
wheelCollider.transform.rotation * Quaternion.Euler(wheelRotation,
wheelCollider.steerAngle, wheelCollider.transform.rotation.z);

        }

        public void WheelCamber (){

                Vector3 wheelLocalEuler;

                if(wheelCollider.transform.localPosition.x < 0)
                        wheelLocalEuler = new
Vector3(wheelCollider.transform.localEulerAngles.x,
wheelCollider.transform.localEulerAngles.y, (-camber));
                else
                        wheelLocalEuler = new
Vector3(wheelCollider.transform.localEulerAngles.x,
wheelCollider.transform.localEulerAngles.y, (camber));

                Quaternion wheelCamber =
Quaternion.Euler(wheelLocalEuler);
                wheelCollider.transform.localRotation =
wheelCamber;

        }

        void SkidMarks(){

                WheelHit GroundHit;
                wheelCollider.GetGroundHit(out GroundHit);

                wheelSlipAmountSideways =
Mathf.Abs(GroundHit.sidewaysSlip);
                wheelSlipAmountForward =
Mathf.Abs(GroundHit.forwardSlip);
                totalSlip = (wheelSlipAmountSideways / 1f) +
(wheelSlipAmountForward / 1f);

                if(skidmarks){

                        if (wheelSlipAmountSideways >
startSlipValue || wheelSlipAmountForward > startSlipValue * 2f){

                                Vector3 skidPoint =
GroundHit.point + 2f * (rigid.velocity) * Time.deltaTime;


        if(rigid.velocity.magnitude > 1f){
                                        lastSkidmark
= skidmarks.AddSkidMark(skidPoint, GroundHit.normal,
(wheelSlipAmountSideways / 2f) + (wheelSlipAmountForward / 2f),
lastSkidmark);

        wheelTemparature += ((wheelSlipAmountSideways / 2f)
+ (wheelSlipAmountForward / 2f)) / ((Time.fixedDeltaTime * 100f)
* Mathf.Lerp(1f, 5f, wheelTemparature / 150f));
                                        }else{
                                                lastSkidmark
= -1;

        wheelTemparature -= Time.fixedDeltaTime * 5f;
                                        }

                                }else{

                                        lastSkidmark = -1;
```

```
                                        wheelTemparature -= Time.fixedDeltaTime * 5f;
                                    }
                                    wheelTemparature = Mathf.Clamp(wheelTemparature, 0f, 150f);
                                }
                            }
                            void Frictions(){
                                WheelHit GroundHit;
                                wheelCollider.GetGroundHit(out GroundHit);
                                bool contacted = false;
                                for (int i = 0; i < physicsFrictions.Length; i++) {
                                    if(GroundHit.point != Vector3.zero && GroundHit.collider.sharedMaterial == physicsFrictions[i].groundMaterial){
                                        contacted = true;
                                        forwardFrictionCurve.stiffness = physicsFrictions[i].forwardStiffness;
                                        sidewaysFrictionCurve.stiffness = (physicsFrictions[i].sidewaysStiffness * tractionHelpedSidewaysStiffness);
                                        wheelCollider.forwardFriction = forwardFrictionCurve;
                                        wheelCollider.sidewaysFriction = sidewaysFrictionCurve;
                                        wheelCollider.wheelDampingRate = physicsFrictions[i].damp;
                                        if (!RCC_Settings.Instance.dontUseAnyParticleEffects)
                                            emission = allWheelParticles[i].emission;
                                        audioClip = physicsFrictions[i].groundSound;
                                        if (wheelSlipAmountSideways > physicsFrictions[i].slip || wheelSlipAmountForward > physicsFrictions[i].slip * 2f){
                                            emission.enabled = true;
                                        }else{
                                            emission.enabled = false;
                                        }
                                    }
                                }
                                if(!contacted && physicsMaterials.useTerrainSplatMapForGroundFrictions){
                                    for (int k = 0; k < physicsMaterials.terrainSplatMapIndex.Length; k++) {
                                        if(GroundHit.point != Vector3.zero && GroundHit.collider.sharedMaterial == physicsMaterials.terrainPhysicMaterial){
                                            if(TerrainSurface.GetTextureMix(transform.position) != null && TerrainSurface.GetTextureMix(transform.position)[k] > .5f){
                                                contacted = true;
                                                forwardFrictionCurve.stiffness = physicsFrictions[physicsMaterials.terrainSplatMapIndex[k]].forwardStiffness;
                                                sidewaysFrictionCurve.stiffness = (physicsFrictions[physicsMaterials.terrainSplatMapIndex[k]].sidewaysStiffness * tractionHelpedSidewaysStiffness);
                                                wheelCollider.forwardFriction = forwardFrictionCurve;
                                                wheelCollider.sidewaysFriction = sidewaysFrictionCurve;
                                                wheelCollider.wheelDampingRate = physicsFrictions[physicsMaterials.terrainSplatMapIndex[k]].damp;
                                                if (!RCC_Settings.Instance.dontUseAnyParticleEffects)
                                                    emission = allWheelParticles[physicsMaterials.terrainSplatMapIndex[k]].emission;
                                                audioClip = physicsFrictions[physicsMaterials.terrainSplatMapIndex[k]].groundSound;
                                                if (wheelSlipAmountSideways > physicsFrictions[physicsMaterials.terrainSplatMapIndex[k]].slip || wheelSlipAmountForward > physicsFrictions[physicsMaterials.terrainSplatMapIndex[k]].slip * 2f){
                                                    emission.enabled = true;
                                                }else{
                                                    emission.enabled = false;
                                                }
                                            }
                                        }
                                    }
                                }
                                if(!contacted){
                                    forwardFrictionCurve.stiffness = orgForwardStiffness;
                                    sidewaysFrictionCurve.stiffness = orgSidewaysStiffness * tractionHelpedSidewaysStiffness;
```

```
                wheelCollider.forwardFriction =
forwardFrictionCurve;
                wheelCollider.sidewaysFriction =
sidewaysFrictionCurve;

                wheelCollider.wheelDampingRate
= physicsFrictions[0].damp;

                if
(!RCC_Settings.Instance.dontUseAnyParticleEffects)
                    emission =
allWheelParticles[0].emission;

                audioClip =
physicsFrictions[0].groundSound;

                if (wheelSlipAmountSideways >
physicsFrictions[0].slip || wheelSlipAmountForward >
physicsFrictions[0].slip * 2f){
                    emission.enabled = true;
                }else{
                    emission.enabled = false;
                }

            }

            if (!contacted &&
!RCC_Settings.Instance.dontUseAnyParticleEffects) {

                for (int i = 0; i <
allWheelParticles.Count; i++) {

                    if
(wheelSlipAmountSideways > startSlipValue ||
wheelSlipAmountForward > startSlipValue * 2f) {

                    } else {
                        emission =
allWheelParticles [i].emission;

        emission.enabled = false;
                    }

                }

            }

        }

        void Audio(){


        if(RCC_Settings.Instance.useSharedAudioSources &&
isSkidding())
                return;

            if(totalSlip > startSlipValue){

                if(audioSource.clip != audioClip)
                    audioSource.clip =
audioClip;

                if(!audioSource.isPlaying)
                    audioSource.Play();

                if(rigid.velocity.magnitude > 1f){
                    audioSource.volume =
Mathf.Lerp(audioSource.volume, Mathf.Lerp(0f, 1f, totalSlip -
startSlipValue), Time.deltaTime * 5f);
                    audioSource.pitch =
Mathf.Lerp(1f, .9f, audioSource.volume);
                }else{
                    audioSource.volume =
Mathf.Lerp(audioSource.volume, 0f, Time.deltaTime * 5f);
                }

            }else{

                audioSource.volume =
Mathf.Lerp(audioSource.volume, 0f, Time.deltaTime * 5f);

                if(audioSource.volume <= .05f)
                    audioSource.Stop();

            }

        }

        bool isSkidding(){

            for (int i = 0; i < allWheelColliders.Count; i++)
{
                if(allWheelColliders[i].totalSlip >
totalSlip)
                    return true;

            }

            return false;

        }

}
```

HR_GamePlayHandler

```
using UnityEngine;
using UnityEngine.UI;
using System;
using System.Collections;
using UnityEngine.SceneManagement;
using DG.Tweening;
using UnityEngine.Analytics;
using GoogleMobileAds.Api;
public class HR_GamePlayHandler : MonoBehaviour {

    public AdInterstatial _adInterstatial;
    public GameObject[] playerCars;
    public GameObject playerCar;
    public GameObject countDown;
    public AudioSource completeSound;
    public AdBanner _adBanner;
    HR_PlayerHandler _handler;
    #region SINGLETON PATTERN
    public static HR_GamePlayHandler _instance;
    public static HR_GamePlayHandler Instance
    {
        get
        {
            if (_instance == null){
                _instance =
GameObject.FindObjectOfType<HR_GamePlayHandler>();
            }

            return _instance;
        }
    }
    #endregion
    [Header("Time Of The Scene")]
    public DayOrNight dayOrNight;
    public enum DayOrNight{Day, Night, Rainy, Snowy}

    [Header("Current Mode")]
```

```
	internal Mode mode;
	internal enum Mode {OneWay, TwoWay, TimeAttack,
Bomb}

	[Header("UI Canvases For GamePlay and GameOver")]
	public Canvas gameplayCanvas;
	public Canvas gameoverCanvas;
	public GameObject[] gameOverAnimBtns;
	public GameObject completeDialog;

	public GameObject camera;

	[Header("Spawn Location Of The Cars")]
	public Transform spawnLocation;

	public  GameObject player;

	private int selectedCarIndex = 0;
	private int selectedModeIndex = 0;
	private float minimumSpeed = 20f;

	internal bool gameStarted = false;
	internal bool paused = false;

	public int totalPlayedCount = 0;

	public delegate void onPlayerSpawned(GameObject p);
	public static event onPlayerSpawned OnPlayerSpawned;

	public delegate void onPlayerEnabled();
	public static event onPlayerEnabled OnPlayerEnabled;

	public delegate void onPlayerDisabled();
	public static event onPlayerDisabled OnPlayerDisabled;

	public delegate void onPlayerCrashed();
	public static event onPlayerCrashed OnPlayerCrashed;

	public delegate void onPaused();
	public static event onPaused OnPaused;

	public delegate void onResumed();
	public static event onResumed OnResumed;
	Image NosImage;
	CarController carController;
  bool SpeedRefresh = true;
  public static bool ShareButtons = false,isCheckPoint;
	public static int highwayNumber;
	public GameObject startDemoPanel,levelsChallange;
	public int checkpointIndex;
	public LevelsData levelDescription;
	Coroutine highwayStatsRoutine=null;

  void Awake(){

		SpawnCar();
			Time.timeScale = 1;
			Scriptt.levelDone = false;
			if (MenuScript.day) {
				dayOrNight = DayOrNight.Day;
			}
			else {
				dayOrNight = DayOrNight.Night;
			}

	if(HR_HighwayRacerProperties.Instance.gameplayClips
!= null &&
HR_HighwayRacerProperties.Instance.gameplayClips.Length > 0)

	RCC_CreateAudioSource.NewAudioSource(gameObject,
"GamePlay Soundtrack", 0f, 0f, .35f,
HR_HighwayRacerProperties.Instance.gameplayClips[UnityEngine.
Random.Range(0,
HR_HighwayRacerProperties.Instance.gameplayClips.Length)], true,
true, false);

			selectedCarIndex =
PlayerPrefs.GetInt("SelectedPlayerCarIndex",0);
			selectedModeIndex =
PlayerPrefs.GetInt("SelectedModeIndex",0);
			totalPlayedCount =
PlayerPrefs.GetInt("TotalPlayedCount",0);

			switch(selectedModeIndex){
			case 0:
				mode = Mode.OneWay;
				break;
			case 1:
				mode = Mode.TwoWay;
				break;
			case 2:
				mode = Mode.TimeAttack;
				break;
			case 3:
				mode = Mode.Bomb;
				break;
			}
//			highwayNumber = 2;
			if(startDemoPanel)
				startDemoPanel.SetActive (true);
			if (highwayNumber > 0) {
				isCheckPoint = true;
				levelsChallange.SetActive (false);
			} else {
				levelsChallange.SetActive (true);
				if (LevelChallange._level == 9) {
					isCheckPoint = false; //
me change
				}
				else {
					isCheckPoint = false;
				}
			}
		}
		public void reSetTrafficMode()
		{
			selectedModeIndex =
PlayerPrefs.GetInt("SelectedModeIndex",0);
			switch(selectedModeIndex){
			case 0:
				mode = Mode.OneWay;
				break;
			case 1:
				mode = Mode.TwoWay;
				break;
			case 2:
				mode = Mode.TimeAttack;
				break;
			case 3:
				mode = Mode.Bomb;
				break;
			}
		}
		void Start () {
			gameplayCanvas.enabled = true;
			gameoverCanvas.enabled = false;
			Screen.sleepTimeout =
SleepTimeout.NeverSleep;


			StartCoroutine(WaitForGameStart());
			if(GameObject.FindGameObjectWithTag
("NosImage"))
				NosImage =
```

```
GameObject.FindGameObjectWithTag
("NosImage").GetComponent<Image> ();
                        CollectableControl.isShield = false;
    ShareButtons = false;
                        checkpointIndex = 1;
  }

        IEnumerator WaitForGameStart(){
                        yield return new WaitForSeconds(6);
                        gameStarted = true;
                        startDemoPanel.SetActive (false);
    CarController.GameSatart = true;
    //CarController.canControl = true;
   //CarController.ins.rb.velocity = new Vector3(0, 0, 40);
  }
        IEnumerator setHighWayPlayerpref()
        {
                                yield return new WaitForSeconds
(1f);
                                if (PlayerPrefs.GetFloat
("HighWay" + highwayNumber + "Stats") < _handler.distance) {
                                        if (PlayerPrefs.GetFloat
("HighWay" + highwayNumber + "Stats") <
levelDescription.highwayDistance [highwayNumber - 1]) {

        PlayerPrefs.SetFloat (("HighWay" + highwayNumber +
"Stats"), _handler.distance);
                                        }

                                }
                                if (highwayStatsRoutine != null)
                                        highwayStatsRoutine =
StartCoroutine (setHighWayPlayerpref ());
        }
        public bool checkHighwayDistanceComplete()
        {
                if (highwayNumber > 0) {
                                if (PlayerPrefs.GetFloat
("HighWay" + highwayNumber + "Stats") >=
levelDescription.highwayDistance [highwayNumber - 1]) {
                                        stopCar ();

        General.Instance.highwayCompleted ();
                                        StopCoroutine
(highwayStatsRoutine);
                                        highwayStatsRoutine =
null;
                                        PlayerPrefs.SetFloat
(("HighWay" + highwayNumber +
"Stats"),(levelDescription.highwayDistance [highwayNumber - 1]));

        if(!PlayerPrefs.HasKey("HighwayAchievement"+highwa
yNumber))
                                        {

        if(highwayNumber==1)

        General.Instance.callAchievement (16); //highway
achievement starts from 16-19

        PlayerPrefs.SetInt("HighwayAchievement"+highwayNum
ber,1);
                                        }
                                        return true;
                                }
                        }
                        return false;
        }
        public void stopCar()
        {
                carController.stopOnComplete ();
    if
(player.GetComponent<CarStats>().shieldParticle.activeInHierarchy)

player.GetComponent<CarStats>().shieldParticle.SetActive(false);
    if
(player.GetComponent<CarStats>().shieldTruck.activeInHierarchy)
    {

player.GetComponent<CarStats>().shieldTruck.transform.GetChild(
0).gameObject.SetActive(false);

player.GetComponent<CarStats>().shieldTruck.transform.GetChild(
2).gameObject.SetActive(false);

player.GetComponent<CarStats>().shieldTruck.transform.GetChild(
3).gameObject.SetActive(false);
    }
  }
        public void countDownAnim()
        {
                countDown.SetActive (false);
                countDown.SetActive (true);
        }
  public void SpawnCar () {

                if (mode != Mode.Bomb) {
                        player = (GameObject)Instantiate
(playerCars [MenuScript.selectedCar]);
                        player.SetActive (false);
                }
                player.transform.position =
spawnLocation.transform.position;
                OnPlayerSpawned (player);
                player.transform.rotation =
Quaternion.identity;
                carController =
player.GetComponent<CarController> ();
                player.SetActive (true);
                player.GetComponent<CarStats>
().setCarControllerValues ();
                SmoothFollow.target=player.transform;
                SmoothFollow.hoodCam =
player.transform.GetChild(0).transform;
                _handler =
player.GetComponent<HR_PlayerHandler> ();
                if (highwayNumber > 0) {
                        highwayStatsRoutine =
StartCoroutine (setHighWayPlayerpref ());
                }
        }

        public IEnumerator OnGameOver(float delayTime){
                _adBanner.DisableBanner();
                _adInterstatial.ShowAd();
                if (General.levelComplete) {
                        delayTime = 3f;
        Time.timeScale = 1.0f;
        Time.fixedDeltaTime = 0.0167f;
        yield return new WaitForSeconds(2);
        completeDialog.SetActive (true);
                }
                else {
                        delayTime = 2f;
                }
                yield return new WaitForSeconds(delayTime);
                completeDialog.SetActive (false);
                gameplayCanvas.enabled = false;
                OnPaused ();
                gameoverCanvas.enabled = true;
                //foreach (GameObject btn in
gameOverAnimBtns) {
                //
        btn.GetComponent<DOTweenAnimation> ().DORestart
```

```
();
                        //}
      ShareButtons = true;

            GameObject.FindObjectOfType<HR_Scoreboard>().Disp
layResults();
                        switch(mode){

                        case Mode.OneWay:

            PlayerPrefs.SetInt("bestScoreOneWay",
(int)player.GetComponent<HR_PlayerHandler>().score);
                                    break;
                        case Mode.TwoWay:

            PlayerPrefs.SetInt("bestScoreTwoWay",
(int)player.GetComponent<HR_PlayerHandler>().score);
                                    break;
                        case Mode.TimeAttack:

            PlayerPrefs.SetInt("bestScoreTimeAttack",
(int)player.GetComponent<HR_PlayerHandler>().score);
                                    break;
                        case Mode.Bomb:

            PlayerPrefs.SetInt("bestScoreBomb",
(int)player.GetComponent<HR_PlayerHandler>().score);
                                    break;

                        }

                        totalPlayedCount++;
                        PlayerPrefs.SetInt("TotalPlayedCount",
totalPlayedCount);
            }

            void OnLevelWasLoaded(){
                        Time.timeScale = 1;
                        AudioListener.pause = false;
            }

            public void MainMenu()
            {
                        if (Scriptt.levelDone)
                        {

            SceneManager.LoadScene(SceneManager.GetActiveScen
e().buildIndex);

                        }
                        else
                        {
                                    LoadingScript.leveltoload = 1;
                                    General.NextLevelAd = true;

                                    Application.LoadLevel (8);
                        }
            }

            public void RestartGame(){
//      #if !UNITY_EDITOR
//                        if (Ads_Manager.Instance) {
//                                    if (PlayerPrefs.GetInt
("ADSUNLOCK") != 1) {
//
            Ads_Manager.Instance.HideLargeAdmobBanner ();
//
            Ads_Manager.Instance.HideLargeAdmobBanner1 ();
//                                    }
//                        }
//#endif
                        if (Scriptt.levelDone) { //next level
                                    LoadingScript.leveltoload = 1;
                                    General.NextLevelAd = true;
                                    Application.LoadLevel (1);
                        } else { //restart level

            SceneManager.LoadScene(SceneManager.GetActiveScen
e().buildIndex);

                        }


            }
            public void nextLevel()
            {

                        if (LevelChallange.instance) {
                                    if (LevelChallange._level <
MenuScript.totalLevels) {
                                                LevelChallange._level
+= 1;

            SceneManager.LoadScene
(SceneManager.GetActiveScene ().buildIndex);
                                    }
                                    else {

            LoadingScript.leveltoload = 1;
                                                General.NextLevelAd =
true;
                                                Application.LoadLevel
(1);
                                    }
                        }
//#if !UNITY_EDITOR
//                        if (Ads_Manager.Instance) {
//                                    if (PlayerPrefs.GetInt
("ADSUNLOCK") != 1) {
//
            Ads_Manager.Instance.HideLargeAdmobBanner ();
//
            Ads_Manager.Instance.HideLargeAdmobBanner1 ();
//                                    }
//                        }
//#endif
   }
   public void RestartGame2()
   {
                        if (PlayerPrefs.GetInt ("ADSUNLOCK") != 1)
{
//#if !UNITY_EDITOR
//
            Ads_Manager.Instance.HideLargeAdmobBanner ();
//
            Ads_Manager.Instance.HideLargeAdmobBanner1 ();
//#endif
                        }
      if (Scriptt.levelDone) //next level
      {
         LoadingScript.leveltoload = 1;
         General.NextLevelAd = true;
         Application.LoadLevel(1);

      }
      else //restart level
      {
        // SceneManager.LoadScene("11-Levels");

      }

   }
```

```
	public void GotoLevelSelection()
	{
		if (PlayerPrefs.GetInt ("ADSUNLOCK") != 1)
{
//#if !UNITY_EDITOR
//
	Ads_Manager.Instance.HideLargeAdmobBanner ();
//
	Ads_Manager.Instance.HideLargeAdmobBanner1 ();
//#endif
		}

		LoadingScript.leveltoload = 1;
		General.NextLevelAd = true;
		Application.LoadLevel(1);

		PlayerPrefs.SetInt("GotoLevels",1);
	}


	public void Paused(){

		paused = !paused;

		if(paused)
			OnPaused ();
		else
			OnResumed ();

	}

#region free cash
	public GameObject freeCashPanel;
	public void openCashPanel()
	{
		freeCashPanel.SetActive (true);
	}
	public void closeCashPanel()
	{
		General.Instance.playClickSound ();
		freeCashPanel.SetActive (false);
	}
#endregion

}
```

HR_ModificationColor

```
using UnityEngine;
using UnityEngine.UI;
using System;
using System.Collections;
using System.Collections.Generic;

public class HR_ModificationColor : MonoBehaviour {

	public pickedColor _pickedColor;
	public enum pickedColor{Orange, Red, Green, Blue,
Yellow, Black, White, Cyan, Magenta, Pink}
	public int colorPrice;
	public bool unlocked = false;
	private Text priceLabel;
	private Image priceImage;

	void Start(){

		priceLabel =
GetComponentInChildren<Text>();
		priceImage =
priceLabel.GetComponentInParent<Image>();

	}

	public void OnClick () {

		if(!unlocked){
			BuyColor();
			return;
		}

		HR_ModHandler handler =
GameObject.FindObjectOfType<HR_ModHandler>();
		Color selectedColor = new Color();

		switch(_pickedColor){

		case pickedColor.Orange:
			selectedColor = Color.red +
(Color.green / 2f);
			break;

		case pickedColor.Red:
			selectedColor = Color.red;
			break;

		case pickedColor.Green:
			selectedColor = Color.green;
			break;

		case pickedColor.Blue:
			selectedColor = Color.blue;
			break;

		case pickedColor.Yellow:
			selectedColor = Color.yellow;
			break;

		case pickedColor.Black:
			selectedColor = Color.black;
			break;

		case pickedColor.White:
			selectedColor = Color.white;
			break;

		case pickedColor.Cyan:
			selectedColor = Color.cyan;
			break;

		case pickedColor.Magenta:
			selectedColor = Color.magenta;
			break;

		case pickedColor.Pink:
			selectedColor = new Color(1, 0f,
.5f);
			break;

		}

		handler.ChangeChassisColor(selectedColor);

	}

	void Update(){

		string currentColorString =
_pickedColor.ToString();

		if(colorPrice <= 0 && !unlocked){
```

```
		PlayerPrefs.SetInt(GameObject.FindObjectOfType<RCC
_CarControllerV3>().transform.name + "OwnedColor" +
currentColorString, 1);
				unlocked = true;
			}


		if(PlayerPrefs.HasKey(GameObject.FindObjectOfType<
RCC_CarControllerV3>().transform.name + "OwnedColor" +
currentColorString))
				unlocked = true;
			else
				unlocked = false;

			if(!unlocked){

		if(!priceImage.gameObject.activeSelf)

		priceImage.gameObject.SetActive(true);
				if(priceLabel.text !=
colorPrice.ToString())
					priceLabel.text =
colorPrice.ToString();
			}else{

		if(priceImage.gameObject.activeSelf)

		priceImage.gameObject.SetActive(false);
				if(priceLabel.text !=
"UNLOCKED")
					priceLabel.text =
"UNLOCKED";
			}

		}

		void BuyColor(){

			int playerCoins =
PlayerPrefs.GetInt("Currency");
			string currentColorString =
_pickedColor.ToString();

			if(playerCoins >= colorPrice){
				HR_ModHandler handler =
GameObject.FindObjectOfType<HR_ModHandler>();
				handler.BuyProperty(colorPrice,
GameObject.FindObjectOfType<RCC_CarControllerV3>().transfor
m.name + "OwnedColor" + currentColorString);
			}

		}

}
```

HR_ModificationUpgrade

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class HR_ModificationUpgrade : MonoBehaviour {

		public UpgradeClass upgradeClass;
		public enum UpgradeClass{Speed, Handling, Brake,
Siren}
		public HR_ModApplier applier;
		public int upgradePrice;
		public bool fullyUpgraded = false;
		public Text priceLabel;
		private Image priceImage;

		void Awake () {

			priceImage =
priceLabel.GetComponentInParent<Image>();

		}

		void OnEnable(){

			applier =
GameObject.FindObjectOfType<HR_ModApplier>();

		}

		public void OnClick () {

			int playerCoins =
PlayerPrefs.GetInt("Currency");

			if(fullyUpgraded && upgradeClass ==
UpgradeClass.Siren){

		GameObject.FindObjectOfType<HR_ModHandler>().Up
gradeSiren();
			}

			if(playerCoins < upgradePrice)
				return;

			if(!fullyUpgraded)
				BuyUpgrade();

		}

		void Update(){

			switch(upgradeClass){

			case UpgradeClass.Speed:
				if(applier.speedLevel >= 5){
					fullyUpgraded = true;
				}else{
					fullyUpgraded = false;
				}
				break;
			case UpgradeClass.Handling:
				if(applier.handlingLevel >= 5){
					fullyUpgraded = true;
				}else{
					fullyUpgraded = false;
				}
				break;
			case UpgradeClass.Brake:
				if(applier.brakeLevel >= 5){
					fullyUpgraded = true;
				}else{
					fullyUpgraded = false;
				}
				break;
			case UpgradeClass.Siren:
				if(applier.isSirenPurchased ==
true){
					fullyUpgraded = true;
				}else{
					fullyUpgraded = false;
				}
				break;

			}

			if(!fullyUpgraded){
```

```
            if(!priceImage.gameObject.activeSelf)

            priceImage.gameObject.SetActive(true);
                        if(priceLabel.text !=
upgradePrice.ToString())
                                    priceLabel.text =
upgradePrice.ToString();
                        }else{

            if(priceImage.gameObject.activeSelf)

            priceImage.gameObject.SetActive(false);
                        if(priceLabel.text !=
"UPGRADED")
                                    priceLabel.text =
"UPGRADED";
                        }

            }

            void BuyUpgrade(){

                        int playerCoins =
PlayerPrefs.GetInt("Currency");
                        HR_ModApplier applier =
GameObject.FindObjectOfType<HR_ModApplier>();
                        HR_ModHandler handler =
GameObject.FindObjectOfType<HR_ModHandler>();

                        if(playerCoins >= upgradePrice){

                                    switch(upgradeClass){

                                    case UpgradeClass.Speed:
                                                if(applier.speedLevel <
5){

            handler.UpgradeSpeed();

            PlayerPrefs.SetInt("Currency", playerCoins -
upgradePrice);
                                                }
                                                break;
                                    case UpgradeClass.Handling:
                                                if(applier.handlingLevel
< 5){

            handler.UpgradeHandling();

            PlayerPrefs.SetInt("Currency", playerCoins -
upgradePrice);
                                                }
                                                break;
                                    case UpgradeClass.Brake:
                                                if(applier.brakeLevel <
5){

            handler.UpgradeBrake();

            PlayerPrefs.SetInt("Currency", playerCoins -
upgradePrice);
                                                }
                                                break;
                                    case UpgradeClass.Siren:

            if(applier.isSirenPurchased == false){

            handler.UpgradeSiren();

            PlayerPrefs.SetInt("Currency", playerCoins -
upgradePrice);
                                                }
                                                break;

                                    }
                        }
            }
}
```

HR_ModificationWheel

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class HR_ModificationWheel : MonoBehaviour {

            public int wheelIndex;
            public int wheelPrice{get{return
HR_Wheels.Instance.wheels[wheelIndex].price;}}
            public bool unlocked;
            private Text priceLabel;
            private Image priceImage;

            void Start () {

                        priceLabel =
GetComponentInChildren<Text>();
                        priceImage =
priceLabel.GetComponentInParent<Image>();
                        unlocked =
HR_Wheels.Instance.wheels[wheelIndex].unlocked;

            }

            public void OnClick () {

                        if(!unlocked){
                                    BuyWheel();
                                    return;
                        }

                        HR_ModHandler handler =
GameObject.FindObjectOfType<HR_ModHandler>();

                        handler.ChangeWheels(wheelIndex);

            }

            void Update(){

                        string currentWheelString =
wheelIndex.ToString();

                        if(wheelPrice <= 0 && !unlocked){

            PlayerPrefs.SetInt(GameObject.FindObjectOfType<RCC
_CarControllerV3>().transform.name + "OwnedWheel" +
currentWheelString, 1);
                                    unlocked = true;
                        }


            if(PlayerPrefs.HasKey(GameObject.FindObjectOfType<
RCC_CarControllerV3>().transform.name + "OwnedWheel" +
currentWheelString) ||
HR_Wheels.Instance.wheels[wheelIndex].unlocked)
                                    unlocked = true;
                        else
```

```
                                unlocked = false;

                        if(!unlocked){

            if(!priceImage.gameObject.activeSelf)

            priceImage.gameObject.SetActive(true);
                                if(priceLabel.text !=
wheelPrice.ToString())
                                        priceLabel.text =
wheelPrice.ToString();
                        }else{

            if(priceImage.gameObject.activeSelf)

            priceImage.gameObject.SetActive(false);
                                if(priceLabel.text !=
"UNLOCKED")
                                        priceLabel.text =
"UNLOCKED";
                        }

            }



void BuyWheel(){

                        int playerCoins =
PlayerPrefs.GetInt("Currency");
                        string currentWheelString =
wheelIndex.ToString();

                        if(playerCoins >= wheelPrice){
                                HR_ModHandler handler =
GameObject.FindObjectOfType<HR_ModHandler>();
                                handler.BuyProperty(wheelPrice,
GameObject.FindObjectOfType<RCC_CarControllerV3>().transfor
m.name + "OwnedWheel" + currentWheelString);
                        }

            }

}
```

HR_PlayerCars

```
using UnityEngine;
using System.Collections;

[System.Serializable]
public class HR_PlayerCars : ScriptableObject {

            public static HR_PlayerCars instance;
            public static HR_PlayerCars Instance
            {
                        get
                        {
                                    if(instance == null)
                                                instance =
Resources.Load("HR_Assets/HR_PlayerCars") as HR_PlayerCars;
                                    return instance;
                        }

            }

            [Space(10f)]

            public GameObject bombedVehicleForBombMode;

            [System.Serializable]
            public class Cars{
                        public GameObject playerCar;

                        public float maxSpeed;
                        public float maxHandling;
                        public float maxBraking;

                        public bool unlocked = false;
                        public int price = 25000;

            }

            public Cars[] cars;

}
```

HR_PlayerHandler

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
using UnityEngine.Analytics;

[RequireComponent (typeof(Rigidbody))]
//[RequireComponent (typeof(RCC_CarControllerV3))]
[RequireComponent (typeof(HR_ModApplier))]
public class HR_PlayerHandler : MonoBehaviour {

   public static HR_PlayerHandler Instance;
            private CarController carController;

            private bool gameOver = false;
            private bool gameStarted{get{return
HR_GamePlayHandler.Instance.gameStarted;}}

            public float score;
            internal float timeLeft = 100f;
            internal int combo;
            internal int wrongWayScore;
            internal int maxCombo;

            internal float speed = 0f,speedOriginal=0f;
            internal float distance = 0f;
            internal float highSpeedCurrent = 0f;
            internal float highSpeedTotal = 0f;
            internal float opposideDirectionCurrent = 0f;
            internal float opposideDirectionTotal = 0f;
            [HideInInspector] public float levelTime;
   bool fist = true;
            int temp = 0;
   int Collide = 1000000000;
   private int minimumSpeedForGainScore
            {
                        get
                        {
                                    return
HR_HighwayRacerProperties.Instance._minimumSpeedForGainScor
e;
                        }
            }
            private int minimumSpeedForHighSpeed
            {
                        get
                        {
                                    return
HR_HighwayRacerProperties.Instance._minimumSpeedForHighSpe
ed;
                        }
            }

            private Vector3 previousPosition;
```

```
	private string currentTrafficCarNameLeft;
	private string currentTrafficCarNameRight;

	public GameObject nearMissText;
	public int nearMisses;
	public static float comboTime;

	internal bool bombTriggered = false;
	internal float bombHealth = 100f;
	public float timer_WrongWay=0;

	public bool Isonce;


	void Awake () {

//
	if(!GameObject.FindObjectOfType<RCC_UIDashboardDisplay>()){
//		enabled = false;
//		return;
//	}
		carController = GetComponentInParent<CarController>();
		nearMissText = GameObject.FindGameObjectWithTag ("NearMiss");
		nearMissText.SetActive(false);
    Instance = this;
  }
	void Start()
	{
		speedRoutine=null;
    previousPosition = transform.position;
		Isonce = false;
		wrongWayScore = 0;
		combo = 0;
		maxCombo = 0;

	}
	void ChangeLanes()
	{

		HR_TrafficCar[] cars = FindObjectsOfType<HR_TrafficCar> ();
		for (int i = 0; i < cars.Length; i++) {
			if (!cars [i].changingLanes) {
				cars [i].ChangeLanes ();
				temp++;
			}
		}
		if (temp >= 15) {
			fist = false;
		}
  }
	Coroutine speedRoutine;
	IEnumerator normalizeSpeed()
	{
		if (carController.currentSpeed < 100) {
			speed = 50;
		} else if (carController.currentSpeed > 100 && carController.currentSpeed <= 800) {
//			speed = Random.Range (80, 80);
			//speed = 70;
		} else if (carController.currentSpeed > 800 && carController.currentSpeed <= 1200) {
			speed = 100;
		} else if (carController.currentSpeed > 1200 && carController.currentSpeed <= 1600) {
			speed = 110;
		} else if (carController.currentSpeed > 1600 && carController.currentSpeed <= 2000) {
			speed = 120;
		} else if (carController.currentSpeed > 2000 && carController.currentSpeed <= 2400) {
			speed = 130;
		} else if (carController.currentSpeed > 2400 && carController.currentSpeed <= 2800) {
			speed = 140;
		} else if (carController.currentSpeed > 2800 && carController.currentSpeed <= 3200) {
			speed = 150;
		} else if (carController.currentSpeed > 3200 && carController.currentSpeed <= 3600){
			speed = 160;
		}
		else if (carController.currentSpeed > 3600 && carController.currentSpeed <= 4000){
			speed = 170;
		}
		else if (carController.currentSpeed > 4000 && carController.currentSpeed <= 4400){
			speed = 180;
		}
		else if (carController.currentSpeed > 4400 && carController.currentSpeed <= 4800){
			speed = 190;
		}
		else if (carController.currentSpeed > 5200 && carController.currentSpeed <= 5600){
			speed = 200;
		}
		else if (carController.currentSpeed > 5600 && carController.currentSpeed <= 6000){
			speed = 210;
		}
		else{
			speed = 220;
		}
		yield return new WaitForSeconds (0.3f);
		speedRoutine = null;
	}
	void Update () {

    if (gameOver || !gameStarted)
			return;
		if (CarController.canControl) {
			if(speedRoutine==null)
			{
//			speed =carController.currentSpeed;
			//
        speedRoutine=StartCoroutine(normalizeSpeed());
			}
			speedOriginal = carController.currentSpeed;
//			normalizeSpeed();
			distance += Vector3.Distance (previousPosition, transform.position) / 2200f;
			previousPosition = transform.position;
			if (distance >= 6.8f && fist)
				ChangeLanes ();
      if (speedOriginal >= minimumSpeedForGainScore && LevelChallange._level < 13)//nos score ad in collect level therefore tis _level condition apply
      {
        if (LevelChallange.instance)
        {
          if (LevelChallange.instance.isWrongWayLevel)
          {
            if (LevelChallange.instance.isWrongWay)
            {
              score += carController.currentSpeed * (Time.deltaTime * .15f) * 1.5f;
            }
```

```
                }
                else
                {
                    score += carController.currentSpeed *
(Time.deltaTime * .15f) * 1.5f;
                }
            }
            else
            {
                score += carController.currentSpeed * (Time.deltaTime
* .15f) * 1.5f;
            }
        }

        if (speedOriginal >= minimumSpeedForHighSpeed) {
                                        highSpeedCurrent +=
Time.deltaTime;
                                        highSpeedTotal +=
Time.deltaTime;
                                } else {
                                        highSpeedCurrent = 0f;
                                }

                                if (/*speedOriginal >=
(minimumSpeedForHighSpeed / 2f) &&*/transform.position.x <= 0f
&& HR_GamePlayHandler.Instance.mode ==
HR_GamePlayHandler.Mode.TwoWay) {
                                        if
(LevelChallange.instance) {

            LevelChallange.instance.isWrongWay = true;
                                        }
                                        timer_WrongWay +=
Time.deltaTime;


            opposideDirectionCurrent += Time.deltaTime;
                                        opposideDirectionTotal
+= Time.deltaTime;


                                        //timer_WrongWay =
opposideDirectionCurrent;

                                        //print(timer_WrongWay
+" timer ");

                                        if (timer_WrongWay >=
10 && !Isonce )
                                        {

            wrongWayScore ++;


            //print(wrongWayScore +" Dinero ");
                                                Isonce = true;
                                        }
                                        else if(Isonce)
                                        {
                                                Isonce =
false;

            timer_WrongWay = 0;
                                        }


                                } else {

            opposideDirectionCurrent = 0f;
                                        if
(LevelChallange.instance) {

            LevelChallange.instance.isWrongWay = false;
                                        }
                                        Isonce = false;
                                        //timer_WrongWay = 0;
                                }

                                if
(HR_GamePlayHandler.Instance.mode ==
HR_GamePlayHandler.Mode.TimeAttack) {
                                        timeLeft -=
Time.deltaTime;
                                        if (timeLeft < 0) {
                                                timeLeft = 0;
                                                OnGameOver
(0f);
                                        }
                                }

                                comboTime += Time.deltaTime;

                                if
(HR_GamePlayHandler.Instance.mode ==
HR_GamePlayHandler.Mode.Bomb) {

                                        if (speedOriginal > 80f)
{
                                                if
(!bombTriggered)

            bombTriggered = true;
                                                else

            bombHealth += Time.deltaTime * 5f;
                                        } else if
(bombTriggered) {
                                                bombHealth -
= Time.deltaTime * 10f;
                                        }

                                        bombHealth =
Mathf.Clamp (bombHealth, 0f, 100f);

                                        if (bombHealth <= 0f) {
                                                GameObject
explosion = (GameObject)Instantiate
(HR_HighwayRacerProperties.Instance.explosionEffect,
transform.position, transform.rotation);

            explosion.transform.SetParent (null);

            GetComponent<Rigidbody> ().isKinematic = true;
                                                OnGameOver
(2f);
                                        }
                                }
                        }
                        if(comboTime >= 5)
                                combo = 0;

                        if (General.levelComplete) {
                                OnGameOver (1f);
                        }

        }

        public void AnimateNearMiss()
        {
                nearMisses ++;
                combo ++;
                comboTime = 0;
                if(maxCombo <= combo)
                        maxCombo = combo;
```

```
            RCC_CreateAudioSource.NewAudioSource(gameObject,
HR_HighwayRacerProperties.Instance.nearMissAudioClip.name, 0f,
0f, 1f, HR_HighwayRacerProperties.Instance.nearMissAudioClip,
false, true, true);

                        nearMissText.SetActive(true);

            nearMissText.GetComponentInChildren<Animator>().Pla
y(0);
                        if (LevelChallange.instance) {
                                    if
(LevelChallange.instance.isWrongWayLevel) {
                                                if
(LevelChallange.instance.isWrongWay) {
                  score += (100f * Mathf.Clamp(combo, 1f, 200f));
                  print(" WRONG WAY"+ score);
              }
                                    } else {
                                                score += 100f *
Mathf.Clamp (combo, 1f, 200f);
          }

      } else {
                                    score += 100f * Mathf.Clamp
(combo, 1f, 200f);
      }
     // print("near missobject name = "+ nearMissText);

            nearMissText.transform.GetChild(0).GetComponent<Tex
t>().text = "Risk +"+ (100f * Mathf.Clamp(combo, 1f,
200f)).ToString("F0");
      if (combo == 50 && PlayerPrefs.GetInt("Achievement20")!=1)
      {

            General.Instance.callAchievement(20);

            PlayerPrefs.SetInt("Achievement20", 1);
      }
      if (combo == 100 && PlayerPrefs.GetInt("Achievement21") !=
1)
      {

            General.Instance.callAchievement(21);

            PlayerPrefs.SetInt("Achievement21", 1);
      }
   }

            void CheckNearMiss(){

                        RaycastHit hit;


            if(Physics.Raycast(carController.centreofmass.position, (-
transform.right), out hit, 2f,
HR_HighwayRacerProperties.Instance.trafficCarsLayer) &&
!hit.collider.isTrigger){
                                    currentTrafficCarNameLeft =
hit.transform.name;
                        }else{

                                    if(currentTrafficCarNameLeft !=
null && speedOriginal >
HR_HighwayRacerProperties.Instance._minimumSpeedForGainScor
e){

                                                nearMisses ++;
                                                combo ++;
                                                comboTime = 0;
                                                if(maxCombo <=
combo)
                                                            maxCombo =
combo;


            RCC_CreateAudioSource.NewAudioSource(gameObject,
HR_HighwayRacerProperties.Instance.nearMissAudioClip.name, 0f,
0f, 1f, HR_HighwayRacerProperties.Instance.nearMissAudioClip,
false, true, true);


            nearMissText.SetActive(true);

            nearMissText.GetComponentInChildren<Animator>().Pla
y(0);
                                    if
(LevelChallange.instance) {
                                                if
(LevelChallange.instance.isWrongWayLevel) {
                                                            if
(LevelChallange.instance.isWrongWay) {

            score += 100f * Mathf.Clamp(combo, 1f, 20f);
Debug.Log ("13");
                                                            }
                                                } else {

            score += 100f * Mathf.Clamp(combo, 1f, 20f);
Debug.Log ("12");
                                                }
                        } else {
                                                score += 100f
* Mathf.Clamp(combo, 1f, 20f); Debug.Log ("11");
            }


            nearMissText.GetComponentInChildren<Text>().text =
"Risk +" + (100f * Mathf.Clamp(combo , 1f, 20f)).ToString("F0");


            currentTrafficCarNameLeft = null;

                                    }else{


            currentTrafficCarNameLeft = null;

                                    }

                        }


            if(Physics.Raycast(carController.centreofmass.position,
(transform.right), out hit, 2f,
HR_HighwayRacerProperties.Instance.trafficCarsLayer) &&
!hit.collider.isTrigger){
                                    currentTrafficCarNameRight =
hit.transform.name;
                        }else{

                                    if(currentTrafficCarNameRight !=
null && speedOriginal >
HR_HighwayRacerProperties.Instance._minimumSpeedForGainScor
e){

                                                nearMisses ++;
                                                combo ++;
                                                comboTime = 0;
                                                if(maxCombo <=
combo)
                                                            maxCombo =
combo;
```

```
            RCC_CreateAudioSource.NewAudioSource(gameObject,
HR_HighwayRacerProperties.Instance.nearMissAudioClip.name, 0f,
0f, 1f, HR_HighwayRacerProperties.Instance.nearMissAudioClip,
false, true, true);


            nearMissText.SetActive(true);

            nearMissText.GetComponentInChildren<Animator>().Pla
y(0);


                                                if
(LevelChallange.instance) {
                                                    if
(LevelChallange.instance.isWrongWayLevel) {
                                                        if
(LevelChallange.instance.isWrongWay) {

            score += 100f * Mathf.Clamp(combo / 1.5f, 1f, 20f);
                                                        }
                                                    } else {

            score += 100f * Mathf.Clamp(combo / 1.5f, 1f, 20f);
                }
            } else {
                                                    score += 100f
* Mathf.Clamp(combo / 1.5f, 1f, 20f);
            }



            nearMissText.GetComponentInChildren<Text>().text =
"Risk +" + (100f * Mathf.Clamp(combo / 1.5f, 1f,
20f)).ToString("F0");


            currentTrafficCarNameRight = null;

                                }else{


            currentTrafficCarNameRight = null;

                                }

                        }


            if(Physics.Raycast(carController.centreofmass.position,
(transform.forward), out hit, 40f,
HR_HighwayRacerProperties.Instance.trafficCarsLayer) &&
!hit.collider.isTrigger){ //change lane on horn
//
//
            if(carController.highBeamHeadLightsOn)
//
            hit.transform.SendMessage("ChangeLines");

                        }

            }

    void Hit()
    {
        Collide = 0;
    }
    public void OnGameOver(float delayTime){
                                fist = false;
                                gameOver = true;
            HR_GamePlayHandler.Instance.StartCoroutine
("OnGameOver", delayTime);
            }

}
```

HR_RoadPooling

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
//using VacuumShaders.CurvedWorld;
public class HR_RoadPooling : MonoBehaviour {

    public GameObject[] CheckPoint;
    public Camera mainCamera;
            public GameObject wrongWayCollider;
            internal Transform reference;
            private bool animateNow = true;

            [System.Serializable]
            public class RoadObjects{
                        public GameObject roadObject;
            }

            public int roadAmountInPool = 10;
            private float[] roadLength;
            [Header("Use This Layer On Road For Calculating Road
Length")] public LayerMask asphaltLayer;

            [Header("Pooling Road Objects. Select Them While They
Are On Your Scene")] public RoadObjects[] roadObjects;
            internal List<GameObject> roads = new
List<GameObject>();

            public float roadWidth = 13.5f;

            public bool automaticRoadLength = true;
            public float manualRoadLength = 60f;
            private int index = 0;
            public GameObject roadLl;
            public GameObject curvedController;
            void Awake () {

        reference = mainCamera.transform;
                        roadLength = new float[roadObjects.Length];

                        for (int i = 0; i < roadObjects.Length; i++) {
                                    //roadLength[i] =
GetRoadLength(i);
                                    if(automaticRoadLength)
                                                roadLength[i] =
GetRoadLength(i);
                                    else
                                                roadLength[i] =
manualRoadLength;
                        }
                        Invoke("CreateRoads",0.2f);
            }
            void Start()
            {
                        if (HR_GamePlayHandler.isCheckPoint) {
                                    foreach (GameObject _checkpoint
in CheckPoint) {
                                    //          _checkpoint.SetActive
(true);
                                    }
                        }
                        else {
                                    foreach (GameObject _checkpoint
```

```
in CheckPoint) {
                                        //_checkpoint.SetActive
(false);
                                }
                        }

        }
        protected float GetRoadLength (int roadIndex){

                        GameObject roadReference =
(GameObject)GameObject.Instantiate(roadObjects[roadIndex].roadO
bject, Vector3.zero, Quaternion.identity);

                        Bounds combinedBounds =
roadReference.GetComponentInChildren<Renderer>().bounds;
                        Renderer[] renderers =
roadReference.GetComponentsInChildren<Renderer>();

                        foreach (Renderer render in renderers) {
                                if (render !=
roadReference.GetComponent<Renderer>() && 1 <<
render.gameObject.layer == asphaltLayer)

        combinedBounds.Encapsulate(render.bounds);
                        }

                        Destroy(roadReference);
                        return combinedBounds.size.z;

        }

        void CreateRoads () {

                        GameObject allRoads = new GameObject("All
Roads");
                        allRoads.transform.position = Vector3.zero;
                        allRoads.transform.rotation =
Quaternion.identity;

                        for (int i = 0; i < roadAmountInPool; i++) {

                                for (int k = 0; k <
roadObjects.Length; k++) {

                                        GameObject go =
(GameObject)GameObject.Instantiate(roadObjects[k].roadObject,
roadObjects[k].roadObject.transform.position,
roadObjects[k].roadObject.transform.rotation);
                                        go.isStatic = false;
                                        roads.Add(go);

        HR_SetLightmapsManually.Instance.AlignLightmaps(roa
dObjects[k].roadObject, go);

        go.transform.SetParent(allRoads.transform);
                                }

                        }

                        for (int i = 0; i < roads.Count; i++) {

                                if(i != 0)

        roads[i].transform.position = new Vector3(0f, 0, roads[i -
1].transform.position.z + roadLength[(index <= 0) ?
roadObjects.Length - 1 : index - 1]);

                                index ++;

                                if(index >= roadObjects.Length)
                                        index = 0;
                        }

                        for (int j = 0; j < roadObjects.Length; j++) {

        if(roadObjects[j].roadObject.activeSelf)

        roadObjects[j].roadObject.SetActive(false);

                        }

                        index = 0;

        }

        void Update(){

                        if(animateNow)
                                AnimateRoads();
        }

        void AnimateRoads () {

                        for (int i = 0; i < roads.Count; i++) {

                                if(reference.transform.position.z >
(roads[i].transform.position.z + (roadLength[index] * 2f))){

        roads[i].transform.position = new Vector3(0f, 0,
(roads[i].transform.position.z + (roadLength[index] * roads.Count)));
                                }
                                index ++;

                                if(index >= roadObjects.Length)
                                        index = 0;

                        }

        }
        int currentRoad,i;
        public static bool isChangeCurveRoad;
        Transform referenceNext;
        void assignRows()
        {
                        isChangeCurveRoad = false;
        //
        curvedController.GetComponent<CurvedWorld_Controll
er> ().pivotPoint = roads [currentRoad].transform;
                        if (currentRoad < roadAmountInPool - 1) {
                                currentRoad++;
                        } else if (currentRoad == roadAmountInPool-1
) {
                                currentRoad = 0;
                        }

        }

        void OnDrawGizmos(){

                        Gizmos.color = new Color(0f, 1f, 0f, .75f);
                        if(roadLl)
                        Gizmos.DrawCube(new
Vector3(roadLl.transform.position.x,0f,0f), new Vector3(roadWidth
* 3f, 1f, 10f));
                        else
                                Gizmos.DrawCube(Vector3.zero,
new Vector3(roadWidth * 3f, 1f, 10f));
        }

}
```

HR_Scoreboard

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
using UnityEngine.Advertisements;

public class HR_Scoreboard : MonoBehaviour {

	public GameObject doubleMoneyButton;
	public GameObject doubleScoreLabel;

	[Header("UI Texts On Scoreboard")]
	public Text totalScore;
	public Text subTotalMoney;
	public Text totalMoney;
	//public static bool endSound = true;
	public Text totalDistance;
	public Text totalNearMiss;
	public Text totalOverspeed;
	public Text totalOppositeDirection;
	public Text totalPuzzle;
	public Text totalDistanceMoney;
	public Text totalNearMissMoney;
	public Text totalOverspeedMoney;
	public Text totalOppositeDirectionMoney;
	public Text totalPuzzleMoney;
	public Text comboCount;
	public Text TotalCombo;
	public Text
TotalOppositeDineros,TotalOppositeCountDinero,TotalDineros;

	public int totalDineroScore;

	bool heyzaponce;
	bool hello;
	int puzzleReward;
	int scoretemp;
	int tempGold;
	public static AudioSource DingSound;
   public bool isVideoWatched = false;
   public GameObject AdButton;
   public void Start()
	{
		puzzleReward = 100;
		//endSound = true;
		DingSound =
GameObject.FindGameObjectWithTag
("Menu").GetComponent<AudioSource>();
	}

	public void Update()
	{
		if (PlayerPrefs.GetInt ("SHARE") == 1) {
			totalMoney.text = (tempGold +
tempGold).ToString ();
			PlayerPrefs.SetInt
("Currency",PlayerPrefs.GetInt("Currency") + tempGold +
tempGold);
			tempGold += tempGold;
			PlayerPrefs.SetInt ("SHARE", 0);
		}
		if (isVideoWatched == true)
      {
         isVideoWatched = false;

         totalMoney.text = (Mathf.Floor(tempGold +
tempGold)).ToString ();
         PlayerPrefs.SetInt
("Currency",PlayerPrefs.GetInt("Currency") + tempGold );
			tempGold += tempGold;
		}
	}


	public int totalDistanceMoneyMP
	{
		get
		{
			return
HR_HighwayRacerProperties.Instance._totalDistanceMoneyMP;
		}
	}
	public int totalNearMissMoneyMP{
		get
		{
			return
HR_HighwayRacerProperties.Instance._totalNearMissMoneyMP;
		}
	}
	public int totalOverspeedMoneyMP{
		get
		{
			return
HR_HighwayRacerProperties.Instance._totalOverspeedMoneyMP;
		}
	}
	public int totalOppositeDirectionMP{
		get
		{
			return
HR_HighwayRacerProperties.Instance._totalOppositeDirectionMP;
		}
	}
	int puzzleCharacters;
	public void DisplayResults(){

		DingSound.Play ();
		HR_PlayerHandler playerHandler =
GameObject.FindObjectOfType<HR_PlayerHandler>();
      if (playerHandler != null)
      {
         totalScore.text =
Mathf.Floor(playerHandler.score).ToString("F0");
         totalDistance.text = (playerHandler.distance).ToString("F2");
         totalNearMiss.text =
(playerHandler.nearMisses).ToString("F0");
         totalOverspeed.text =
(playerHandler.highSpeedTotal).ToString("F1");
         totalOppositeDirection.text =
(playerHandler.opposideDirectionTotal).ToString("F1");
      }
		if (LevelChallange.instance) {
         if (LevelChallange.DailyEvent > 0)
			puzzleCharacters =
LevelChallange.instance.totalPuzzleCharacters;// meChange
      } else {
			puzzleCharacters = 0;
		}
		totalPuzzle.text = puzzleCharacters.ToString
();
      totalDistanceMoney.text = Mathf.Floor(playerHandler.distance
* totalDistanceMoneyMP).ToString("F0");
		totalNearMissMoney.text =
Mathf.Floor(playerHandler.nearMisses *
totalNearMissMoneyMP).ToString("F0");
		totalOverspeedMoney.text =
Mathf.Floor(playerHandler.highSpeedTotal *
totalOverspeedMoneyMP).ToString("F0");
		totalOppositeDirectionMoney.text =
Mathf.Floor(playerHandler.opposideDirectionTotal *
totalOppositeDirectionMP).ToString("F0");
		if (LevelChallange.instance) {
```

```
        totalPuzzleMoney.text = (puzzleCharacters *
puzzleReward).ToString ();
                        }
                        else {
                                totalPuzzleMoney.text =
totalPuzzle.text = puzzleCharacters.ToString ();
        }

                        tempGold = Mathf.FloorToInt((Mathf.Floor
(playerHandler.distance * totalDistanceMoneyMP) +
(playerHandler.nearMisses * totalNearMissMoneyMP) +
Mathf.Floor (playerHandler.highSpeedTotal *
totalOverspeedMoneyMP) + Mathf.Floor
(playerHandler.opposideDirectionTotal *
totalOppositeDirectionMP))+(puzzleCharacters*puzzleReward));
                        totalMoney.text =
(Mathf.Floor(playerHandler.distance * totalDistanceMoneyMP) +
(playerHandler.nearMisses * totalNearMissMoneyMP) +
Mathf.Floor(playerHandler.highSpeedTotal *
totalOverspeedMoneyMP) +
Mathf.Floor(playerHandler.opposideDirectionTotal *
totalOppositeDirectionMP)+(puzzleCharacters*puzzleReward)).ToSt
ring("F0");
                        PlayerPrefs.SetInt("Currency",
PlayerPrefs.GetInt("Currency", 0) +
Mathf.FloorToInt(Mathf.Floor(playerHandler.distance *
totalDistanceMoneyMP) + (playerHandler.nearMisses *
totalNearMissMoneyMP) +
Mathf.Floor(playerHandler.highSpeedTotal *
totalOverspeedMoneyMP)+(puzzleCharacters*puzzleReward)));

                        scoretemp =
Mathf.FloorToInt(Mathf.Floor(playerHandler.distance *
totalDistanceMoneyMP) + (playerHandler.nearMisses *
totalNearMissMoneyMP) +
Mathf.Floor(playerHandler.highSpeedTotal *
totalOverspeedMoneyMP)+(puzzleCharacters*puzzleReward));

                        comboCount.text =
HR_PlayerHandler.Instance.maxCombo.ToString("F0");
                        TotalCombo.text =
HR_PlayerHandler.Instance.maxCombo.ToString("F0");

                        TotalOppositeDineros.text =
HR_PlayerHandler.Instance.wrongWayScore.ToString("F0");
                        TotalOppositeCountDinero.text =
HR_PlayerHandler.Instance.wrongWayScore.ToString("F0");

                        totalDineroScore =
HR_PlayerHandler.Instance.maxCombo +
HR_PlayerHandler.Instance.wrongWayScore;

                        TotalDineros.text =
totalDineroScore.ToString("F0");

            PlayerPrefs.SetInt("combo",PlayerPrefs.GetInt("combo")
+totalDineroScore);
                        //              comboCount.text =
PlayerPrefs.GetInt("combo").ToString("F0");

                        if (PlayerPrefs.GetInt ("DOUBLESCORE") ==
1) {
                                PlayerPrefs.SetInt("Currency",
PlayerPrefs.GetInt("Currency", 0) +
Mathf.FloorToInt(Mathf.Floor(playerHandler.distance *
totalDistanceMoneyMP) + (playerHandler.nearMisses *
totalNearMissMoneyMP) +
Mathf.Floor(playerHandler.highSpeedTotal *
totalOverspeedMoneyMP)));
                                doubleMoneyButton.SetActive
(false);
                        }
                                PlayerPrefs.SetInt ("TEMPSCORE",
scoretemp);

                                gameObject.BroadcastMessage("Animate");

            gameObject.BroadcastMessage("GetNumber");

                                playerHandler.gameObject.SetActive (false);

            }

            public void ViewVideoAd()
            {
//#if !UNITY_EDITOR

      // ShowOptions options = new ShowOptions();
        //  options.resultCallback = HandleShowResult;

          //if (Advertisement.IsReady("rewardedVideo"))
          //{
          //    Advertisement.Show("rewardedVideo", options);
          //}
      }

//      private void HandleShowResult(ShowResult result)
//      {
//          switch (result)
//          {
//              case ShowResult.Finished:
//                  isVideoWatched = true;
//
            AdButton.GetComponent<Button>().interactable=false;
//                  break;
//              case ShowResult.Skipped:
//                  isVideoWatched = true;
//
            AdButton.GetComponent<Button>().interactable=false;
//                  break;
//              case ShowResult.Failed:
//                  break;
//          }
////#endif
//    }



            public void ShowAD(string zone =
"rewardedVideoZone")
            {
                        //ShowOptions options = new ShowOptions();
                        //options.resultCallback = AdCallbackhanler;
                        //Advertisement.Show (zone, options);
            }

            void endAudio()
            {
                        DingSound.Stop ();
            }

}


HR_TrafficCar


using UnityEngine;
using System.Collections;
//using VacuumShaders.CurvedWorld;

[RequireComponent(typeof(Rigidbody))]
public class HR_TrafficCar : MonoBehaviour
```

```
{

    private Rigidbody rigid;
    public bool changingLanes = false;
    public bool immobilized = false;
    public BoxCollider bodyCollider;
    internal BoxCollider triggerCollider;

    public ChangingLines changingLines;
    public enum ChangingLines { Straight, Right, Left }
    internal int currentLine = 0;

    public float maximumSpeed = 10f;
    private float _maximumSpeed = 10f;
    private float desiredSpeed;
    public float distance = 0f;
    private Quaternion steeringAngle = Quaternion.identity;

    public Transform[] wheelModels;
    private float wheelRotation = 0f;

    [Header("Just Lights. All of them will work on "NOT Important" Render Mode.")]
    public LensFlare[] headLights;
    public Light[] brakeLights;
    public LensFlare[] signalLights;
    private bool headlightsOn = false;
    private bool brakingOn = false;

    private SignalsOn signalsOn;
    private enum SignalsOn { Off, Right, Left, All }
    private float signalTimer = 0f;
    private float spawnProtection = 0f;
    public bool allowNearMiss = true;
    public bool isAnimating;
    [Space(10)]

    public AudioClip engineSound;
    private AudioSource engineSoundSource;

    void Awake()
    {

        rigid = GetComponent<Rigidbody>();
        rigid.drag = 1f;
        rigid.angularDrag = 4f;
        rigid.maxAngularVelocity = 2.5f;

        Light[] allLights = GetComponentsInChildren<Light>();

        foreach (Light l in allLights)
        {
            l.renderMode = LightRenderMode.ForceVertex;
            l.cullingMask = 0;
        }

        distance = 50f;

        if (!bodyCollider)
        {
            //Debug.LogWarning (transform.name + "is missing collider in HR_TrafficCar script. Select your vehicle collider. Assigning collider automatically now, but it may select wrong collider...");
            bodyCollider = GetComponentInChildren<BoxCollider>();
        }

        GameObject triggerColliderGO = new GameObject("TriggerVolume");
        triggerColliderGO.transform.position = bodyCollider.bounds.center;
        triggerColliderGO.transform.rotation = bodyCollider.transform.rotation;
        triggerColliderGO.transform.SetParent(transform, true);
        triggerColliderGO.transform.localScale = bodyCollider.transform.localScale;
        triggerColliderGO.AddComponent<BoxCollider>();
        triggerColliderGO.GetComponent<BoxCollider>().isTrigger = true;
        triggerColliderGO.GetComponent<BoxCollider>().size = bodyCollider.size;
        triggerColliderGO.GetComponent<BoxCollider>().center = bodyCollider.center;

        triggerCollider = triggerColliderGO.GetComponent<BoxCollider>();
        triggerCollider.size = new Vector3(bodyCollider.size.x * 1.5f, bodyCollider.size.y, bodyCollider.size.z + (bodyCollider.size.z * 3f));
        triggerCollider.center = new Vector3(bodyCollider.center.x, 0f, bodyCollider.center.z + (triggerCollider.size.z / 2f) - (bodyCollider.size.z / 2f));
        if (HR_GamePlayHandler.Instance.dayOrNight == HR_GamePlayHandler.DayOrNight.Night || HR_GamePlayHandler.Instance.dayOrNight == HR_GamePlayHandler.DayOrNight.Rainy)
            headlightsOn = true;
        else
            headlightsOn = false;

        engineSoundSource = RCC_CreateAudioSource.NewAudioSource(gameObject, "Engine Sound", 2f, 5f, 1f, engineSound, true, true, false);
        engineSoundSource.transform.localPosition = new Vector3(engineSoundSource.transform.localPosition.x, GameObject.FindObjectOfType<SmoothFollow>().topHeight, -GameObject.FindObjectOfType<SmoothFollow>().topDistance);
        engineSoundSource.pitch = 1.5f;

        rigid.constraints = RigidbodyConstraints.FreezeRotationX | RigidbodyConstraints.FreezeRotationZ | RigidbodyConstraints.FreezePositionY;

        _maximumSpeed = maximumSpeed;

        Transform[] allTransforms = GetComponentsInChildren<Transform>();

        //foreach (Transform t in allTransforms) {
        //      t.gameObject.layer = (int)Mathf.Log (HR_HighwayRacerProperties.Instance.trafficCarsLayer.value, 2);
        //}

        //                  triggerCollider.gameObject.layer = LayerMask.NameToLayer("TrafficCarVolume");
        triggerCollider.gameObject.layer = LayerMask.NameToLayer("TransparentFX");

    }
    void Start()
    {
        //                  for (int i = 0; i < headLights.Length; i++) {
        //
        //                              headLights [i].renderMode = LightRenderMode.ForceVertex;
        //
        //                  }

        for (int i = 0; i < brakeLights.Length; i++)
        {

            brakeLights[i].renderMode = LightRenderMode.ForceVertex;

        }
```

```
    //                  for (int i = 0; i < signalLights.Length; i++) {

    //                      signalLights[i].renderMode =
LightRenderMode.ForceVertex;
    //                  }

  }

  void Update()
  {
    if (CarController.canControl)
    {
      spawnProtection += Time.deltaTime;

      Lights();
      Wheels();
    }
    if (General.levelComplete || General.gameOver)
    {
      engineSoundSource.volume = 0f;
    }
  }

  void Lights()
  {

    signalTimer += Time.deltaTime;

    for (int i = 0; i < signalLights.Length; i++)
    {

      if (signalsOn == SignalsOn.Off)
      {
        signalLights[i].brightness = 0f;
      }

      if (signalsOn == SignalsOn.Left)
      {
        if (signalTimer >= .5f)
        {
          if (signalLights[i].transform.localPosition.x < 0f)
          {
            signalLights[i].brightness = 0f;
          }
        }
        else
        {
          if (signalLights[i].transform.localPosition.x < 0f)
          {
            signalLights[i].brightness = 0.5f; //1f
          }
        }
        if (signalTimer >= 1f)
          signalTimer = 0f;
      }

      if (signalsOn == SignalsOn.Right)
      {
        if (signalTimer >= .5f)
        {
          if (signalLights[i].transform.localPosition.x > 0f)
          {
            signalLights[i].brightness = 0f;
          }
        }
        else
        {
          if (signalLights[i].transform.localPosition.x > 0f)
          {
            signalLights[i].brightness = 0.5f; //1f
          }
        }
        if (signalTimer >= 1f)
          signalTimer = 0f;
      }

      if (signalsOn == SignalsOn.All)
      {
        if (signalTimer >= .5f)
        {
          signalLights[i].brightness = 0f;
        }
        else
        {
          signalLights[i].brightness = 0.5f; //1f
        }
        if (signalTimer >= 1f)
          signalTimer = 0f;
      }

    }

    for (int i = 0; i < headLights.Length; i++)
    {

      if (!headlightsOn)
        headLights[i].brightness =
Mathf.Lerp(headLights[i].brightness, 0f, Time.deltaTime * 10f);
      else
        headLights[i].brightness =
Mathf.Lerp(headLights[i].brightness, 0.5f, Time.deltaTime * 10f);

    }

    for (int i = 0; i < brakeLights.Length; i++)
    {

      if (brakingOn)
      {
        brakeLights[i].intensity =
Mathf.Lerp(brakeLights[i].intensity, 1f, Time.deltaTime * 10f);
      }
      else
      {
        if (!headlightsOn)
          brakeLights[i].intensity =
Mathf.Lerp(brakeLights[i].intensity, 0f, Time.deltaTime * 10f);
        else
          brakeLights[i].intensity =
Mathf.Lerp(brakeLights[i].intensity, .3f, Time.deltaTime * 10f);
      }

    }

  }

  void Wheels()
  {

    for (int i = 0; i < wheelModels.Length; i++)
    {

      wheelRotation += desiredSpeed * 20 * Time.deltaTime;
      wheelModels[i].transform.localRotation =
Quaternion.Euler(wheelRotation, 0f, 0f);

    }

  }

  void FixedUpdate()
  {

    if (CarController.canControl)
```

```
    {
        if (!immobilized)
        {
            desiredSpeed = Mathf.Clamp(maximumSpeed -
Mathf.Lerp(maximumSpeed, 0f, (distance - 10f) / 50f), 0f,
maximumSpeed);
        }
        else
        {
            desiredSpeed = Mathf.Lerp(desiredSpeed, 0f,
Time.fixedDeltaTime);
        }

        if (distance < 50)
            brakingOn = true;
        else
            brakingOn = false;
        distance = Mathf.Lerp(distance, 50, 0.2f);

        if (!immobilized && HR_GamePlayHandler.Instance.mode
!= HR_GamePlayHandler.Mode.TwoWay)
            transform.rotation = Quaternion.Lerp(transform.rotation,
steeringAngle, Time.fixedDeltaTime * 3f);

        rigid.velocity = Vector3.Lerp(rigid.velocity,
transform.forward * desiredSpeed, Time.fixedDeltaTime * 3f);
        rigid.angularVelocity = Vector3.Lerp(rigid.angularVelocity,
Vector3.zero, Time.fixedDeltaTime * 10f);

        if (!immobilized && HR_GamePlayHandler.Instance.mode
!= HR_GamePlayHandler.Mode.TwoWay)
        {

            switch (changingLines)
            {
                case ChangingLines.Straight:
                    steeringAngle = Quaternion.identity;
                    transform.position = new
Vector3(Mathf.Lerp(transform.position.x,
HR_TrafficPooling.Instance.lines[currentLine].position.x, 0.1f),
transform.position.y, transform.position.z);
                    rigid.velocity = Vector3.Lerp(rigid.velocity,
transform.forward * maximumSpeed, Time.fixedDeltaTime * 3f);
                    break;

                case ChangingLines.Left:

                    if (currentLine == 0)
                    {
                        changingLines = ChangingLines.Straight;
                        break;
                    }
                if (transform.position.x <=
HR_TrafficPooling.Instance.lines[currentLine - 1].position.x + .6f)
                    {
                        currentLine--;
                        signalsOn = SignalsOn.Off;
                        changingLines = ChangingLines.Straight;
                        steeringAngle = Quaternion.identity;
                        //SetSpeed();
                    }
                    else
                    {
                        steeringAngle = Quaternion.identity *
Quaternion.Euler(0f, -5f, 0f);
                        signalsOn = SignalsOn.Left;
                    }
                    break;

                case ChangingLines.Right:

                    if (currentLine ==
(HR_TrafficPooling.Instance.lines.Length - 1))
                    {
                        changingLines = ChangingLines.Straight;
                        break;
                    }

                if (transform.position.x >=
HR_TrafficPooling.Instance.lines[currentLine + 1].position.x - .5f)
                    {
                        currentLine++;
                        signalsOn = SignalsOn.Off;
                        changingLines = ChangingLines.Straight;
                        steeringAngle = Quaternion.identity;
                        //SetSpeed();
                    }
                    else
                    {
                        steeringAngle = Quaternion.identity *
Quaternion.Euler(0f, 5f, 0f);
                        signalsOn = SignalsOn.Right;
                    }
                    break;

            }

        }
    }

    void OnTriggerStay(Collider col)
    {

        if ((1 << col.gameObject.layer) !=
HR_HighwayRacerProperties.Instance.trafficCarsLayer.value ||
col.isTrigger)
            return;

        distance = Vector3.Distance(transform.position,
col.transform.position);

    }

    void OnTriggerExit(Collider col)
    {

        if ((1 << col.gameObject.layer) !=
HR_HighwayRacerProperties.Instance.trafficCarsLayer.value)
            return;

    }

    void OnCollisionEnter(Collision col)
    {
        if (immobilized || spawnProtection < .5f || col.collider.tag !=
"Player")
            return;

        immobilized = true;
        signalsOn = SignalsOn.All;
        InvokeRepeating("immobiliz", 1f, 5f);
    }
    void immobiliz()
    {
        immobilized = false;
    }

    public void OnReAligned()
    {
        //SetSpeed();
        immobilized = false;
        spawnProtection = 0f;
        rigid.velocity = transform.forward * maximumSpeed;
```

```
        rigid.angularVelocity = Vector3.zero;
        signalsOn = SignalsOn.Off;
        changingLines = ChangingLines.Straight;
        distance = 50f;

    }

    void SpeedUp()
    {

        distance = 50f;

    }

    public void ChangeLines()
    {

        if (changingLines == ChangingLines.Left || changingLines ==
ChangingLines.Right)
            return;

        int randomNumber = Random.Range(0, 2);

        changingLines = randomNumber == 0 ? ChangingLines.Left :
ChangingLines.Right;

    }
    void SetSpeed()
    {
        if (MenuScript.selectedCar == 0 && MenuScript.selectedCar
<= 2)
        {
            if (currentLine == 0 || currentLine == 2)
            {
                maximumSpeed = 40;
                //                                      Debug.Log ("fast lane");
            }
            else
            {
                maximumSpeed = 30f;
                //                                      Debug.Log ("slow
lane");
            }
        }
        if (MenuScript.selectedCar >= 3 && MenuScript.selectedCar
<= 5)
        {
 if (currentLine == 0 || currentLine == 2)
{
    maximumSpeed = 45;
//       Debug.Log("fast lane");
            }
            else
            {
 maximumSpeed = 35f;
   //Debug.Log("slow lane");
            }
        }
    if (MenuScript.selectedCar >= 6)
        {
    if (currentLine == 0 || currentLine == 2)
            {
     maximumSpeed = 50;
      //              Debug.Log("fast lane");
            }
      else
            {
                maximumSpeed = 40f;
                 //              Debug.Log("slow lane");
            }
        }
        //else if (transform.position.x < -1.7f && transform.position.x >
-5.2f) {
        //      maximumSpeed = 37.5f;
        //      Debug.Log ("2th lane slow");
        //      } else {
        //      maximumSpeed = 51;
        //      Debug.Log ("1st lane fast");
        //      }
        _maximumSpeed = maximumSpeed;
        //maximumSpeed = Random.Range(_maximumSpeed,
_maximumSpeed * 1.5f);
    }
    public void ChangeLanes()
    {
        changingLanes = true;
        InvokeRepeating("ChangeLines", 10, 25);
    }
}
```

HR_Wheels

```
using UnityEngine;
using System.Collections;

[System.Serializable]
public class HR_Wheels : ScriptableObject {

        public static HR_Wheels instance;
        public static HR_Wheels Instance
        {
                get
                {
                        if(instance == null)
                                instance =
Resources.Load("HR_Assets/HR_Wheels") as HR_Wheels;
                        return instance;
                }

        }

        [System.Serializable]
        public class Wheels{

                public GameObject wheel;
                public bool unlocked;
                public int price;

        }

        public Wheels[] wheels;

}
```