ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА ГРИ В ЖАНРІ 2D ПЛАТФОРМЕР HILL CLIMBING З ВИКОРИСТАННЯМ ДВИГУНА UNITY**»

Виконав: студент 4 курсу, групи ПД-44
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
Савенко В.В.
(прізвище та ініціали)
Керівник Дібрівний О.А.
(прізвище та ініціали)
Рецензент
(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти – «Бакалавр»
Напрям підготовки – 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
О.В. Негоденко
“____” _____ 2022 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Савенко Вадиму Володимировичу
(прізвище, ім’я, по батькові)

1. Тема роботи: Розробка гри в жанрі 2D платформер Hill Climbing з використанням двигуна Unity.

Керівник роботи Дібрівний О.А., доктор філософії, доцент,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від 18 лютого 2022 року №__.

2. Строк подання студентом роботи «3» червня 2022 року
3. Вхідні дані до роботи:
   3.1 Технічна документація
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
   4.1 Розробка комп’ютерної гри у жанрі платформер, використовуючи мову програмування C# за допомогою ігрового рушія Unity та середу розробки Visual Studio для роботи з даною мовою
   4.2 Опис предметної області, визначення функціоналу та алгоритмів проекту
   4.3 Аналіз існуючих ігор у даному жанрі
   4.4 Дослідження рушіїв
   4.5 Визначення функціоналу та алгоритмів проекту

5. Перелік графічного матеріалу:
   5.1 Структурна схема алгоритму

5.2
5.3 Діаграма класів
5.4 Use case діаграама
5.5 Висновки

6. Дата видачі завдання «11» квітня 2022р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підготовка Розділу 1	04.04.2022	
2	Підготовка Розділу 2	04.04.2022-25.04.2022	
4	Підготовка Розділу 3	25.04.2022-06.05.2022	
5	Висновки+Презентація	06.05.2022-13.05.2022	
6	Перевірка роботи на плагіат + Передзахист	16.05.2022-01.06.2022	
7	Захист роботи	02.06.2022-21.06.2022	
8	Випуск	30.06.2022	

Студент _____ Савенко В.В.
(підпис) (прізвище та ініціали)
Керівник роботи _____ Дібрівний О.А
(підпис) (прізвище та ініціали)

Реферат

Текстова частина бакалаврської роботи 57 с., 37 рис., 20 джерел.

ГРА UNITY, 2D РОЗРОБКА, ЖАНР ПЛАТФОРМЕР, РУШІЇ, РЕАЛІЗАЦІЯ ГРИ ЗА ДОПОМОГОЮ МОВИ C#, VISUAL STUDIO.

Об'єкт дослідження – реалізація гри у жанрі Платформер.

Предмет дослідження – гра платформер за допомогою рушія Unity на мові програмування C#.

Мета роботи – аналіз засобів розробки ігор, щоб надалі розробити гру у жанрі Платформер. Створення гри з основними механіками жанру познайомити потенційних гравців з таким цікавим жанром. Отримання цінного досвіду в різних областях, та оволодіти навичками роботи з рушієм, щоб надалі набагато швидше працювати з іншими проектами.

З поставленою метою в проекті вирішили наступні питання:

- Аналіз засобів розробки ігор;
- Аналіз аудиторії;
- Аналіз актуальності продукту;
- Взявши до уваги всі аналізи, розробили гру у жанрі Платформер.

Дана гра дасть змогу гравцеві відчути жанр Платформер, без прикладання великих зусиль розібратись із механіками жанру та вникнути у сюжет.

В роботі виконано аналіз існуючих ігор у жанрі Платформер

Проаналізовано можливість ігрового рушія Unity.

Проаналізовано можливість середовища розробки Visual Studio

Галузь використання – завдяки аналізу, розроблена гра охопить широку аудиторію, яка зацікавлена даним жанром ігор

ЗМІСТ

ВСТУП

Зародження ігрової індустрії відбулося в 1970-х роках, а наразі вона є однією з провідних прибуткових галузей. Сучасна техніка багатьом своїм функціям завдячує ігровій індустрії. Графічні та аудіо карти були розроблені саме для того щоб була можливість інтегровувати музику та зручні інтерфейси користувачів у ігри. CD та DVD диски також отримали розвиток саме завдяки іграм. Розроблені для надійного збереження даних, після початку широкого розповсюдження в ігровій індустрії почалось покращення технологій для швидкості зчитування даних.

Також варто згадати що сучасні ігри є не тільки інструментом для розваг. Існують проекти для підготовки фахівців і різноманітних галузях. Є розвиваючі ігри для дітей та осіб з вадами розвитку.

У нашій країні розвиток ігрової індустрії відбувся дещо пізніше ніж в усьому світі тому ринок компаній які розробляють комп’ютерні ігри не такий великий і є місце для нових студій. Те саме відноситься і до готових ігор власного виробництва. Ніша не є повністю наповненою саме тому було обрано саме цей напрямок.

1 ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ ВІДЕОІГОР

1.1 Що таке відеогра

Відеогра-це комп’ютерна програма, що використовується для створення ігрового процесу, зв’язку з партнерами по грі або сама виступає як партнер по грі. Комп’ютерна гра може бути заснована на базі книг чи фільмів або ж цілком срежесована і зпродюсована геймдизайнером. Процес гри здійснюється завдяки пристроям виводу – монітору, динаміків тощо та пристроїв вводу – клавіатури, мишки, джойстика, гемпада тощо.

1.2 Історія відеоігор

Індустрія відеоігор розпочинається у 1947 році – саме в цьому році було написано першу ігрову програму. Але в звичному нам вигляді відеоігри зародилися у 1960-і роки. Найбільш широкого розповсюдження відеоігри набули в кінці 1970-х завдяки поширенню ігрових автоматів, домашніх приставок та персональних комп’ютерів.

У 1954 після смерті Алана Тьюрінга учені продовжували створення шахів на комп’ютері, проте правила гри були дещо спрощенні через можливості комп’ютерів того часу.

1952 року для електронної обчислювальної машини (EDSAC – Electronic Delay Storage Automatic Calculator), що була створена в 1949 році у Кембридзькому університеті, Александром Дугласом було створено гру «Хрестики-нулики».

У 1961 році в Массачусетському технологічному інституті за допомогою комп’ютера PDP-1 студентами було розроблено гру яка згодом стала всім відома під назвою «Spacewar!».

Першою персональною ігровою приставкою стала Magnavox Odyssey. З'явилася вона у 1972 році і в ній було вбудовано 12 різноманітних ігор.

З початком 1980-х років на території США та Канади був справжній бум відеоігр. Всюди розміщувалися аркадні ігрові автомати та створювалися для них окремі зали. Саме в ті роки вийшли світові хіти «Galaga», «Missile Command», «Pac-Man», «Tempest», «Defender», «Donkey Kong».

У 1990-х роках вже були всесвітньо популярні 2 моделі ігрових приставок – Sega та Nintendo. Завдяки конкуренції цих двох компаній у 1991 році на світ з'явилася гра «Sonic The Hedgehog». Однак одразу головний персонаж гри не був сприйнятий, оскільки в Японії де було створено гру та в США, яка була основним споживачем, багато хто взагалі нічого не знав про їжаків.

Також в саме цей період було створено гру «Mortal Kombat», яка наразі 30 років потому налічує 21 частину.

В середині 1990-х вже почали з'являтись тривимірні ігри. Однією з найперших стала «Tomb Rider».

Вже у 21-му столітті більшість розробників основною аудиторією обирала користувачів персональних комп'ютерів в той час як раніше цільовою аудиторією були власники ігрових приставок або гравці в ігрові автомати. Саме в цей час почав зароджуватись такий жанр відеоігор як MMORPG – Massively multiplayer online role-playing game (Масова багатокористувацька онлайнова рольова гра). Типовим представником цього жанру є «World of Warcraft». Навіть через 20 років після її створення щодня в неї грають більше 10 мільйонів користувачів.

У 2010-х роках світ побачили такі ігри як «Mass Effect 2», «Assassin's Creed», «Metro 2033», «Red Dead Redemption», «Call of Duty» та багато інших відомих ігор. Саме в ці роки з'явилися ігрові консолі 8-го покоління – такі як Xbox One та Sony Play Station 4. В середині десятиліття з'явилися ігри які стали найбільш популярними та касовими у світі – «Grand Theft Auto V», «Counter-Strike: Global Offensive» і «Dota 2».

Саме в цей час з'являється термін «AAA-project». Він означає, що гра орієнтована на масову аудиторію, на розробку та маркетинг якої буде витрачено

більше коштів ніж звичайно. Чітких меж коли гра стає ААА немає, але зазвичай на створення таких ігор раніше витрачали десятки мільйонів доларів, а зараз і сотні.

1.3 Класифікація відеоігор

Основна задача визначення жанру відеоігри залежить від дій гравця. Жанр ігри не залежить від її сюжету та сценарію. Неодноразово різні видання можуть віднести одну й ту саму відеоігру до різних жанрів оскільки єдиної класифікації не розроблено і не існує. Також нерідкість коли в одній грі поєднані декілька різних жанрів і класифікувати її ще важче.

Однак розробники відеоігор зазвичай спираються на усталені поняття про види відеоігор.

Одна з найперших класифікацій відеоігор була розроблена в 1982 році ігровим дизайнером Крісом Кроуфордом в його книзі «The Art of Computer Game Design». Вона виглядала наступним чином:

- Ігри вмінь та дії
- Бойові ігри
- Ігри-лабіринти
- Спортивні ігри
- Відбивання
- Гоночні ігри
- Стратегічні ігри
- Пригоди
- D&D-ігри
- Варгейми
- Ігри шансу
- Навчальні та дитячі ігри
- Міжособистісні ігри
- Різне

1.4 Жанри відеоігор

Основні жанри сучасних відеоігор складаються з багатьох піджанрів:

- Екшн
  - Шутер
  - Файтинг
  - Beat 'em up
  - Платформер
  - Лабіринт
- Стратегія
  - Покрокові стратегічні ігри
  - Стратегічні ігри в реальному часі
    - Tower Defence
    - MOBA (Multiplayer Online Battle Arena)
    - MMORPG (Massively Multiplayer Online Pole-Playing Game)
    - Карткові ігри
    - Глобальні стратегії
- Рольова гра
- Спортивні ігри
  - Командні види спорту
  - Гонки
  - Інші змагання
- Симулятор
- Пригоди
- Інші різновиди
  - Настільна гра

    - Головоломки
        - Вікторини
        - Логічні ігри
    - Games with a Purpose

Також відеоігри класифікуються не тільки за жанрами. Є класифікація:

- За кількістю гравців – один, два чи більше на пристрої
- Мережеві
    - Локальна мережа
    - В мережі інтернет
- За рейтингами видань
- За платформами
- За типом видання
- Та інше

1.5 Складові відеоігор

Будь-яка гра має в собі такі складові:

- Сюжет
- Зміст
- Зовнішнє завдання
- Правила гри
- Засоби гри
- Ризик
- Ігрові дії

1.6 Жанр Платформер

Я обрав жанр платформер оскільки сучасні ігри цього жанру суттєво відрізняються від платформерів минулого століття коли вони зароджувалися.

Вони вже мають у собі безліч інших жанрових належностей, мають продуманий сюжет котрі хоч інколи і був, проте на нього ніколи не звертали належної уваги. Також основною особливістю платформера є варіативність графіки від абсолютно мультиплікаційної до більш-менш реалістичної. То що ж таке платформер?

Жанр гри Платформер відомий також під назвою Платформна гра. Зазвичай ігровий процес відбувається в вигляді руху персонажу по різним платформам, через перешкоди. Впродовж руху відбувається збір різноманітних предметів які і потрібні для завершення рівня.

В класичному платформері рух персонажа відбувається зліва направо. В деяких іграх рухатись можна в будь-якому напрямку а десь рух відбувається автоматично і не залежить від гравця. Під час руху по рівням відбувається збір різних речей необхідних для подальшого проходження гри. Також повсюди розміщенні різноманітні пауер-апи або предмети які надають очки здоров’я або інших використовуємих розхідників.

Також обов’язковим атрибутом цього жанру є різноманітні пастки та противники які керуються штучним інтелектом та заважають гравцю досягти кінця рівня. Знищення таких ворогів зачасту відбувається за рахунок стрибка на них або якщо в грі є зброя то з неї. Знищення ворогів відображується спрощенно – зникнення з екрану або провалювання вниз.

Невід’ємною частиною є також приховані кімнати в які неможливо потрапити при звичному проходженні гри. В таких схованках зазвичай знаходяться предмети які значно прискорюють та полегшують проходження гри і спонукають гравців шукати їх.

Жанровою особливістю також є нереалістичність графіки.

1.7 Перші Платформери

Офіційно першими платформерами був ігри «Frogs», «Space Panic», «Donkey Kong», але у них не було всіх атрибутів «справжнього» платформера.

Першою грою яка мала всі ознаки платформера стала гра «Pitfall!» (1.1).

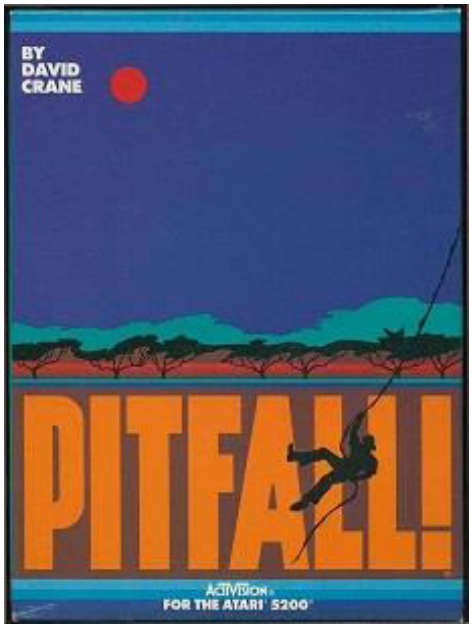


Рис. 1.1 – гра “Pitfall!”

Основною ціллю гри було за відведений час зібрати всі скарби, які розкидані по 255 екранах. В цей час нас нам заважали досягти мети багаточисельні вороги та пастки. Це були різноманітні колоди які знаходилися в стані спокою або рухалися, ями, багаття змії та інше. Всі дії відбувалися в різноманітних тонах сетінгу джунглів.(рис1.2) Однак скролінг екранів відбувався не як в сучасних платформерах поступово, а доходячи до кінця одного екрану одразу вмикався інший.

Рис. 1.2 – Ігровий світ гри “Pitfall”

Справжня популярність до жанру прийшла коли вийшла гра Super Mario Bros. у 1985 році. (рис. 1.3)

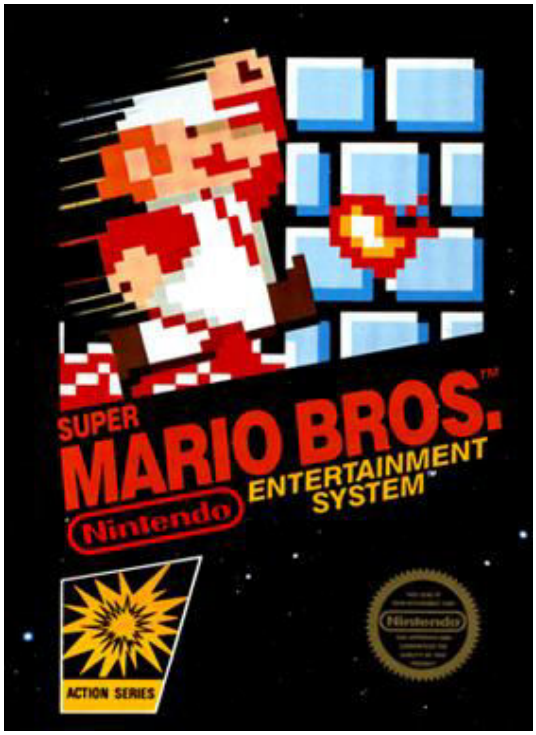


Рис. 1.3 – обкладинка гри “Super Mario Bros”

Випустила її компанія Nintendo для своєї ігрової платформи Famicon. Вона ввійшла до Книги рекордів Гіннеса як гра у якої було продано найбільша кількість копій.

Метою гри було проходження через царство грибів, перестрибуючи через різноманітні пастки і оминати ворогів або знищувати їх (рис. 1.4). Завершенням гри було звільнення полоненої принцеси. Під час проходження рівнів потрібно збирати монети та приховані скриньки. Також на кожному рівні була прихована кімната в якій була збільшена кількість монет та якісь цінні речі що дуже важко знайти.

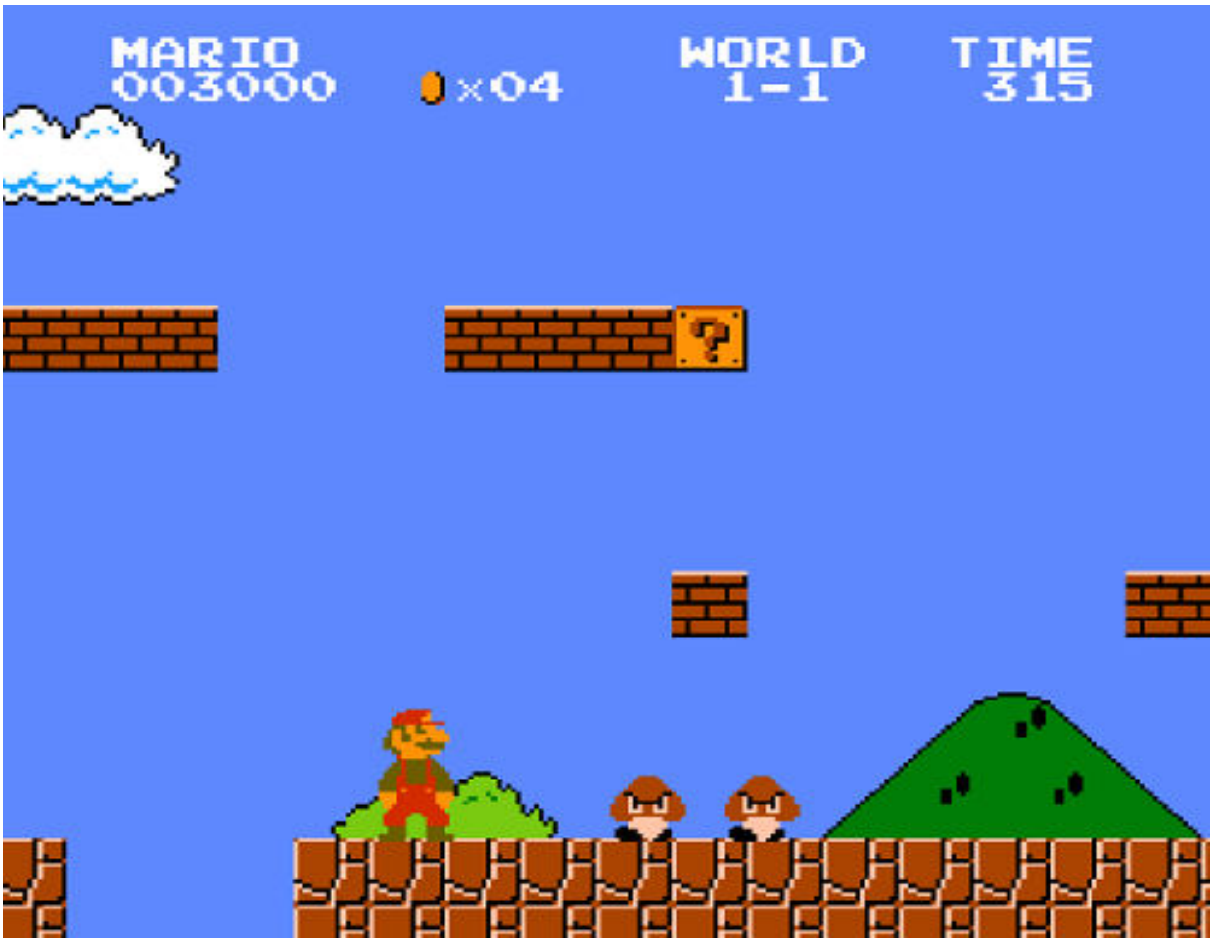


Рис. 1.4 – сетинг гри “Super Mario Bros”

Також однією з ключових ігор становлення жанру стала гра «Sonic the Hedgehog». (рис 1.5) Вона була розроблена компанією Sega для їхньої приставки Sega Mega Drive. Під час виходу гри в неї був самий швидкий геймплей серед тодішніх платформерів і можливості ігрової приставки використовувались на максимум. В основу сюжету було положено керування їжаком схожим на людину якому за відведений час необхідно досягти кінця акту оминаючи різноманітні

перешкоди. Під час проходження рівнів необхідно збирати кільця які потім можна втратити потрапивши в пастку або врізавшись у перепону чи ворога. В подальшому необхідно витрачати кільця на те щоб врятувати острів на якому відбуваються події від антогоніста – Доктора Еггмана.

Рис. 1.5 - гра «Sonic the Hedgehog»

1.8 Сучасні Платформери

Основним атрибутом популярних сучасних платформерів є «розмитість» жанру - їм властиве жанрове розмаїття і часом важко визначити чи є та чи інша гра платформером в класичному розумінні жанру.

Основным атрибутом популярных современных платформеров является размытость жанра – им присущи различные жанровые принадлежности и порой трудно определить является ли та или иная игра платформером в класическом понимании

Одним з сучасних популярних 2D платформерів є гра Hill Climb Racing (рис 1.6). Вона розроблена фінською студією Fingersoft. Випуск відбувся 22 вересня 2012 року. Спочатку реліз відбувся на операційній системі Android, а згодом на Windows, IOS, Windows Phone. Основною метою гравця є як умога далі проїхати нерівною дорогою і в цей час збирати монети та додаткове паливо яке витрачається при русі. Програш настає коли у машини закінчується паливо або вона перевертається та ігровий персонаж травмується в результаті чого не може продовжувати рух.

Рис. 1.6 - гра «Hill Climb Racing»

Earn to Die – ще одна гра жанру платформер. Спочатку вона була Flash-грою, а згодом отримала адаптацію на мобільні пристрої. Механіка гри заключається в постійній модернізації автомобіля задля того щоб проїхати якнайдалі і врешті-решт доїхати до точки евакуації. (рис 1.7)Час від часу на шляху до евакуації можна отримати кращий автомобіль і з його допомогою вдасться проїхати далі. Дуже важливою особливістю гри є те що від сильних пошкоджень при врізанні у пастки автомобіль може развалитися навпіл і шлях доведеться починати заново. Одним з недоліків такого геймплею є постійна монотонність у повторенні одних й тих самих дій, що спонукають гравця до придбання ігрових ресурсів задля пришвидшення прогресу.

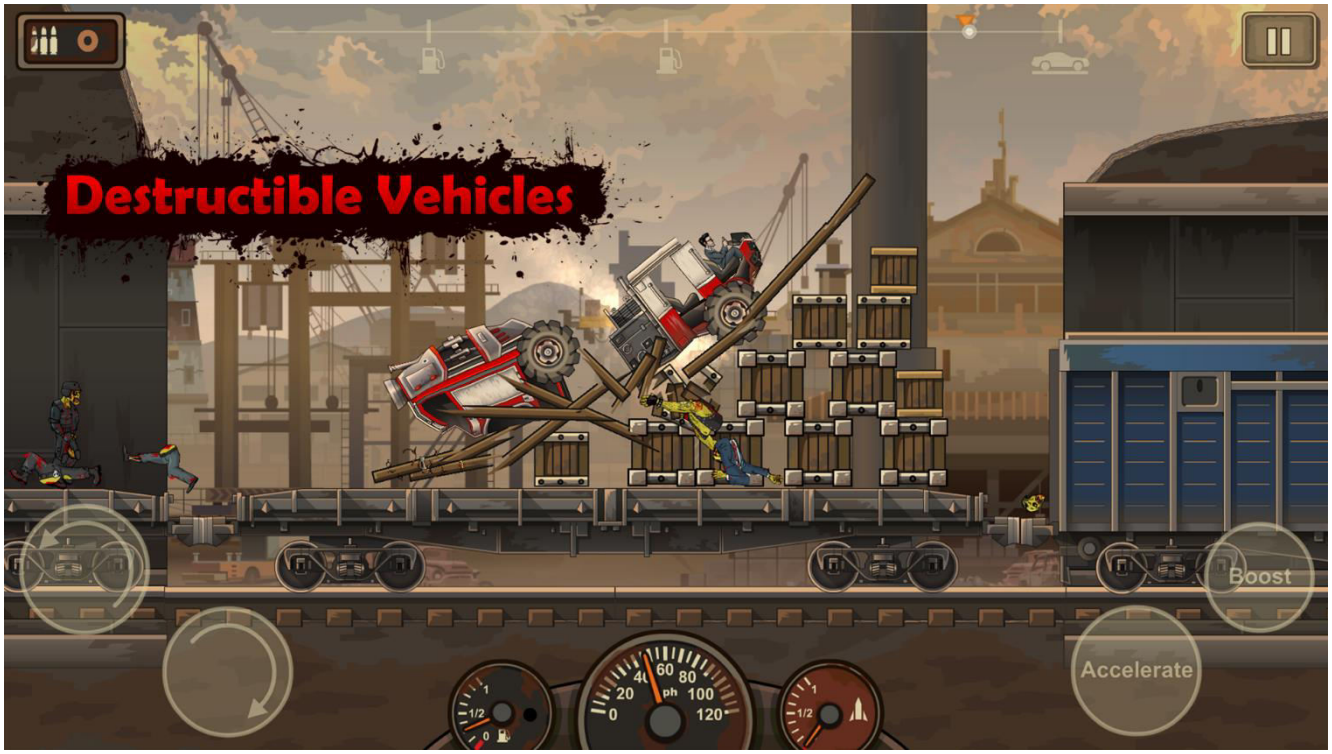


Рис. 1.7 – гра «Earn to Die»

Ще одним представником класичного сучасного платформера виступає гра Gravity Defied: Trial Racing. Вона була розроблена для клавішних телефонів у 2004 році студією Codebrew Software, а з плином десятиліття отримала портування на пристрої з операційною системою Android та IOS. Головною метою

цієї гри є досягти фінішу з найменшим часу проходження рівня оминаючи нерівность рельєфу карти та перестрибуючи величезні прірви.

В основі гейплея полягає три рівні складності в кожному з яких є набір рівнів з різною складністю. Після повного проходження одного рівня складності який у грі називається лігою відкривається доступ до більш потужного мотоциклу який легше керується та має більшу міцність. Однією з найважливіших особливостей гри є те, що всі рівні побудовані методом з’єднання прямих відрізків і немає жодної частини треку на якій є заокруглені місця. (рис. 1.8)

Також варто додати що у гри для того часу є більш-менш реалістична фізика. При стрибках з великої висоти є велика ймовірність зламати мотоцикл. Також швидкість падіння з великих висот не є сталою величиною і постійно пришвидшується.

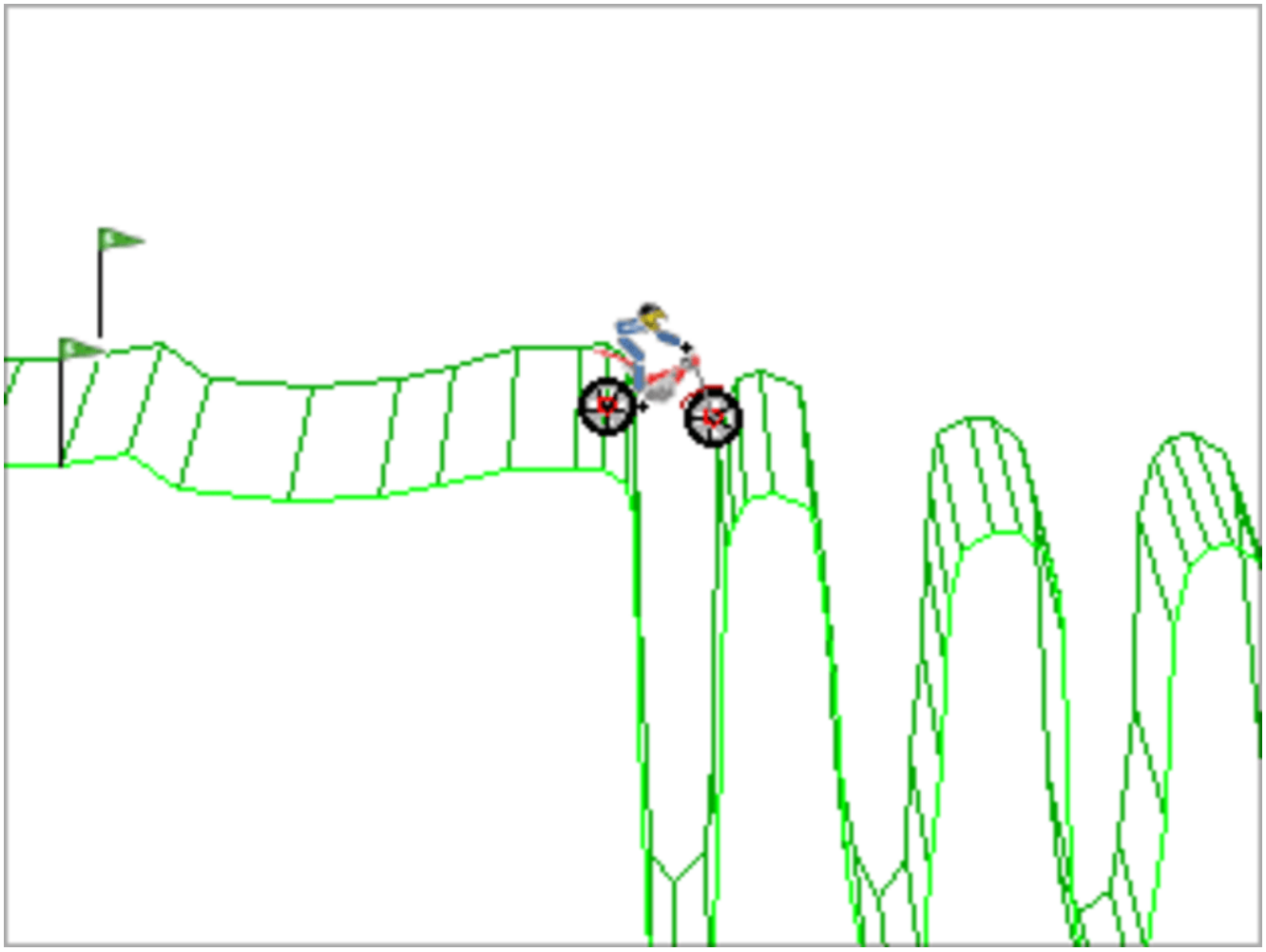


Рис. 1.8 – гра «Gravity Defied: Trial Racing»

Представником класичного платформера є гра Max Fury - Road Warrior Racing. Однак тут вже є ознаки іншого жанру – гонки. В основі гейплею полягає досягти фінішу першим перестрибуючи пастки, які залишають за собою вороги та в цей же час знищувати самих ворогів. В наявності декілька автомобілів кожної категорії – хот-род, спортивні автомобілі, пікапи, вантажівки на які встановлена різноманітна зброя що допоможе знищувати ворога. (рис. 1.9) В процесі гри є можливість покращити характеристики автомобіля та зброї встановленої на авто.

Основою сюжету є постапокаліптичне місто в якому хаос і повністю відсутня влада. Перемагаючи в гонках гравець поступово захоплює владу над містом.

Рис. 1.9 – гра «Max Fury - Road Warrior Racing»

Висновок до розділу 1

Зараз дуже важко знайти гру в якій не буде органічно поєднано декілька сумісних жанрів. Це на краще, адже якщо гра створена виключно в одному жанрі вона орієнтована на вузьке коло користувачів яким цей жанр подобається і скоріш за все гра не отримає великої популярності.

Ціллю будь-якого комерційного проекту є максимально можливий прибуток при помірних інвестиціях. І якщо користувачів багато то прибуток більший. Це дає змогу в подальшому активно розвивати гру та впроваджувати щось нове. У всіх популярних ігрових проектах можна відслідкувати головний жанр в якому створена гра та елементи кількох додаткових жанрів. Це дозволяє розширити коло споживачів що будуть грати, оскільки деякі не є прихильниками основного жанру але їм подобаються суміжні жанри які є в продукті.

2 РОЗРОБКА ПРОЕКТА

2.1 Що таке ігровий рушій

Ігровий рушій – це основа будь-якої гри, в рушії відбуваються всі процеси гри такі як рендер зображення та звуків, виконання скриптів та багато всього іншого. Важливою характеристикою рушія є його мультплатформовість – можливість одночасно обслуговувати персональні комп’ютери, консолі та рідше мобільні пристрої.

Термін ігровий рушій вперше був використаний в середині 1990-х років з появою 3D-ігор. Одними з перших ігор які були створені на базі ігрового рушія стали Quake та Doom. Їх випустила компанія ID Software, яка в подальшому стала продавати свій ігровий рушій іншим компаніям для їх проектів.

Основним завданням ігрових рушіїв є надання комфортних візуальних інструментів для розробки гри. Вони зменшують час який необхідний для надання предмету певних властивостей .

Також ігрові рушії використовують і в непрямому призначенні. Їх використовують при створенні рекламних промороликів, симуляторів для навчання та в архітектурній галузі.

2.2 Аналіз існуючих ігрових рушіїв

2.2.1 Unreal Engine

Unreal Engine – ігровий рушій який створила компанія Epic Games у 1998 році. Самою першою грою що була створена на цьому рушії стала «Unreal». На даний момент актуальна версія двигуна це Unreal Engine 4. Він написан мовою програмування C++, що дає змогу створювати ігри для основних операційних систем таких як Windows, MacOS, Linux та консолей Play Station та Xbox.

Спочатку Unreal Engine розроблювався для шутерів від першої особи, а в подальшому став основою проектів і інших жанрів.

Ігровий двигун постійно вдосконалюється і наразі має зручний інтерфейс для розробників що надає змогу зручно і без додаткових скриптів працювати з об’єктами на сценах. (рис. 2.1)

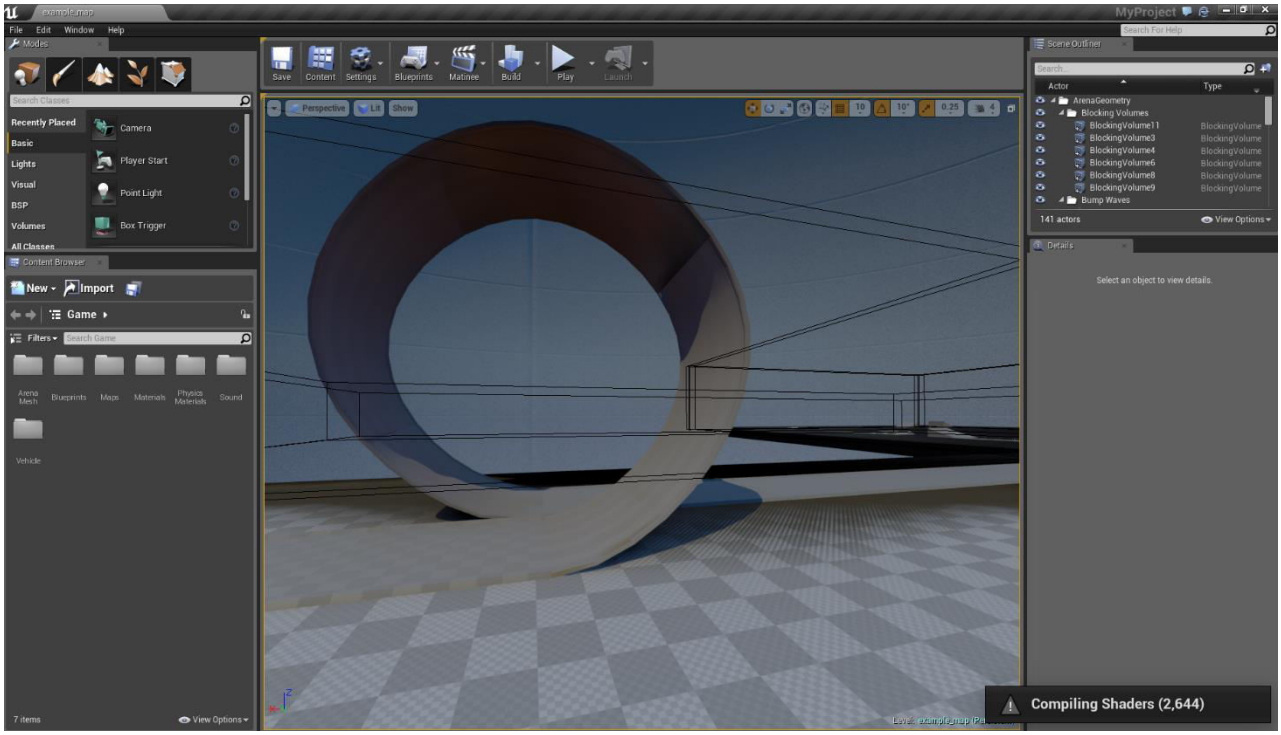


Рис. 2.1 – Інтерфейс рушія Unreal Engine

Більшості ігрових проектів необхідні додаткові скрипти і основною мовою програмування для проектів на Unreal Engine є мова C++.

Ігровий рушій має вбудований редактор візуальних ефектів який допомагає чітко налаштувати реалістичні тіні та відображення. Підтримуються тисячі різних типів поверхонь що ще більше додають реалізму.

Про роботу з рушієм на сайті компанії є структурована інформація для розробників. Є безкоштовний маркет плейс від компанії на якому можна знайти

безкоштовні та платні елементи для розробки – асети, текстури, звуки та багато чого іншого. Він називається Unreal Engine Marketplace.

Двигун Unreal Engine є безкоштовним для некомерційної розробки, але якщо проект розроблений на ньому приносить прибуток то необхідно перераховувати певний відсоток компанії Epic Games.

Unreal Engine розрахований на створення 3D проектів, але має інструменти і для 2D розробників.

2.2.2 Cry Engine

Цей ігровий двигун був створений німецькою компанією Crytek в 2002 році і вперше використаний в грі Far Cry яку побачив світ 23 березня 2004 року.

Основний фурор гра викликала через те що жоден ігровий проект того часу не мав такої детальної графіки (рис. 2.2), не потрібно було додатково завантажувати окремі частини рівнів та весь ігровий світ був доступний одразу. Також варто згадати про те що фізика у грі вийшла на новий рівень і була в більшості моментів абсолютно реалістичною.

Рис. 2.2 – гра «Far Cry»

Наразі всі права власності на ігровий двигун з 2006 року має компанія Ubisoft, яка і надалі розвиває двигун.

У 2016 вийшла остання на даний момент версія рушія, що називається CryEngine V.

Інтерфейс рушія інтуїтивно зрозумілий і приємний на вигляд (рис. 2.3).

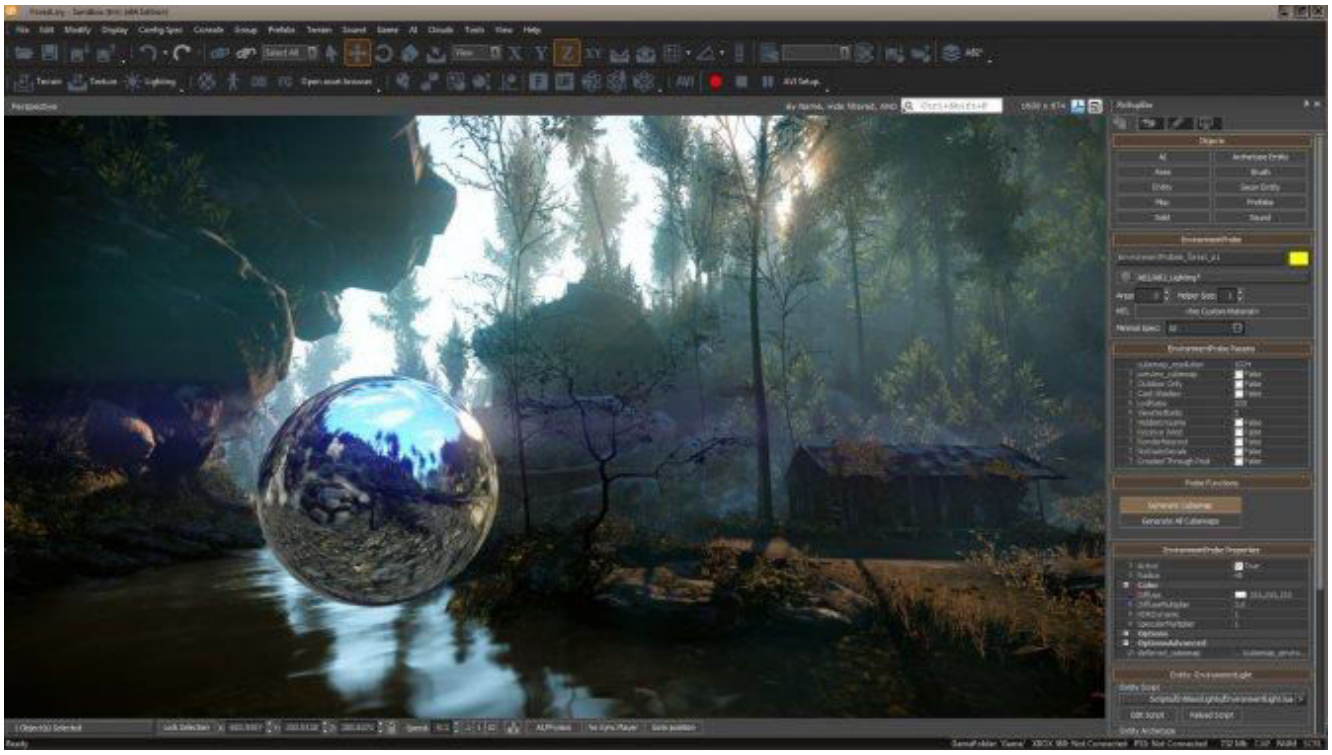


Рис 2.3 – інтерфейс рушія Cry Engine

Основною перевагою цього рушія є прискіплива робота з невеликими деталями навколишньої середи з якими гравець має змогу взаємодіяти. Наприклад, можливість прим'яти траву наступивши на неї.

Завдяки рендеру Vulkan вдається досягти деталізації глобального і локального освітлення та реалізовувати реалістичні тіні від потрапляння на об’єкти світла.

За допомогою вбудованої системи скриптів FlowGraph можливе надання поведінки предметам без написання додаткових скриптів и коду.

CryEngine має вбудовану утиліту Statoscope (рис. 2.4), що дозволяє в режимі реального часу моніторити використання ресурсів системи для подальшої оптимізації проетів.

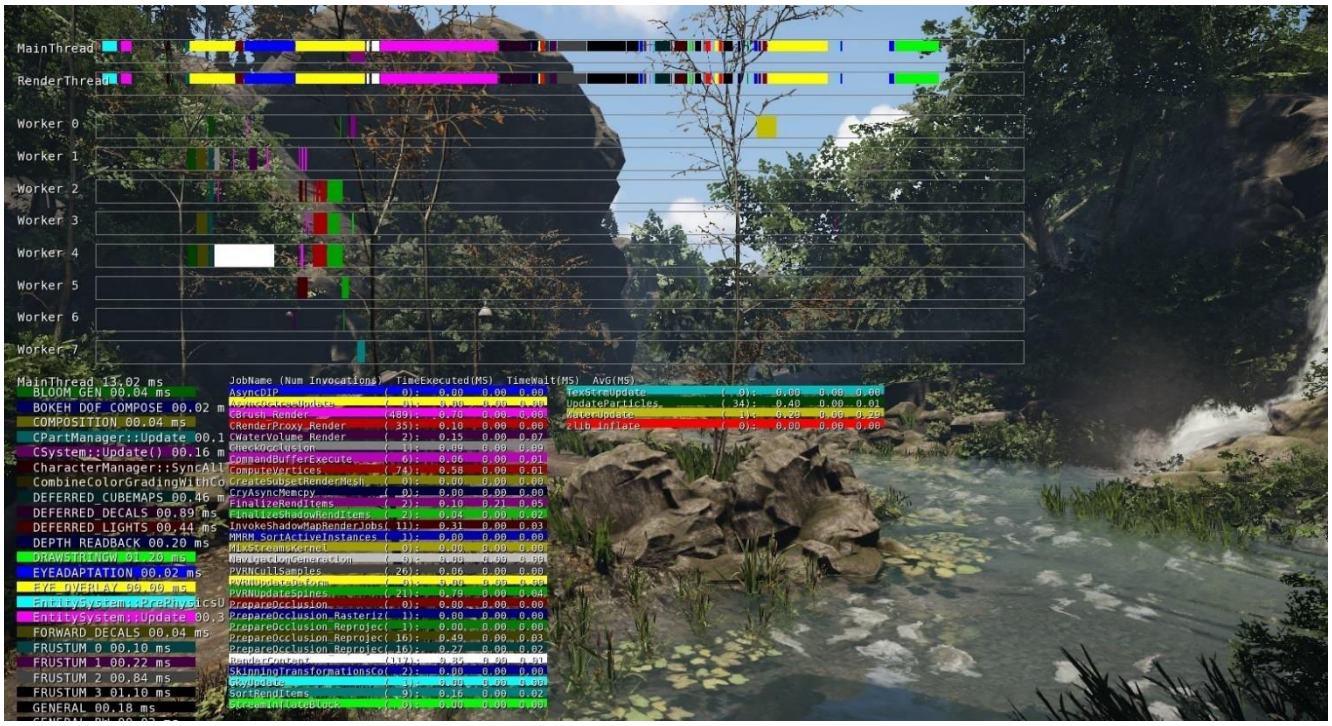


Рис. 2.4 – утиліта «Statoskope»

У рушія є офіційний магазин в якому можна завантажити безкоштовні та платні ресурс паки для власної розробки. Також на сайті двигуна є навчальні матеріали.

З виходом CryEngine 5.5 компанія передивилася політику використання їх рушія. Якщо прибуток від проекту перевищує 5000$, то розробникам необхідно перераховувати 5% прибутку. Але якщо використовується більш рання версія рушія і немає планів в майбутньому переходити на версію 5.5 то можна отримати звільнення від цих виплат.

У підсумку рушій CryEngine є одним з найкращих інструментів для розробки 3D AAA-проектів.

2.2.3 Unity

Unity – це мультиплатформовий інструмент для створення додатків і ігор. Рушій Unity запускається на 25 різних платформах охоплюючи майже всі нині існуючі операційні системи: iOS, Android, Tizen, WindowsUniversal Windows

Platform, Mac, Linux, WebGL, PlayStation 4, PlayStation 5 PlayStation Vita, Xbox One, Xbox Series X, 3DS, Oculus Rift, Google Cardboard, Steam VR, PlayStation VR, Gear VR, Windows Mixed Reality, Daydream, Android TV,Samsung Smart TV, tvOS, Nintendo Switch, Series S, , Facebook Gameroom, Apple ARKit, Google ARCore, Vuforia, і Magic Leap.

Наразі актуальною версією рушія є Unity 2019.

Для розробки на Unity є мінімальні вимоги потужності системи:

- Операційна система: Windows 7 SP1+, 8, 10, 64-bit versions only; Mac OS X 10.12+; Ubuntu 16.04, 18.04, CentOS 7;
- Відеокарта з підтримкою DirectX 10;
- Процесор з підтримкою SSE2.

Інтерфейс для розробників як і у всіх сучасних рушіїв комфортний та зрозумілий (рис. 2.5).

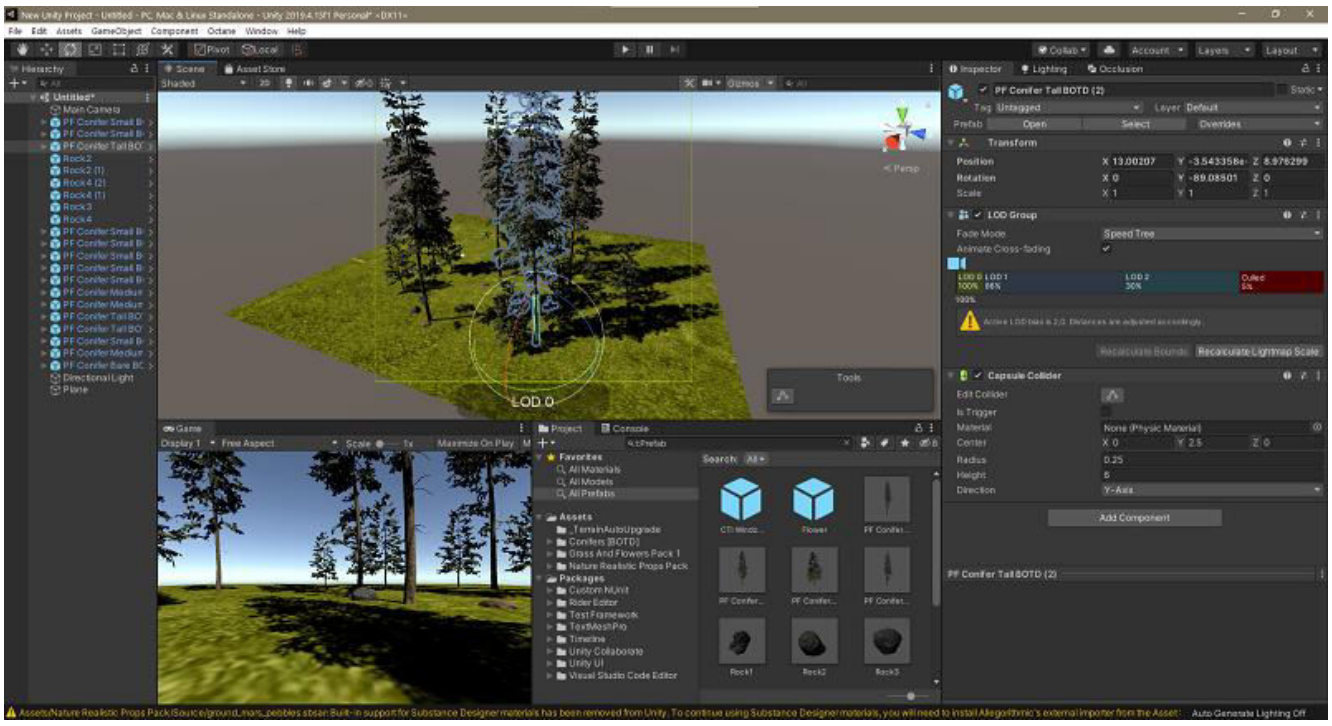


Рис. 2.5 – Інтерфейс ігрового рушія Unity

Ігровий рушій містить у собі різні додаткові інструменти для детальної роботи з різними аспектами розробки:

- Cinemachine – набір інструментів для створення динамічних, «розумних» не потребуючих додаткового програмування камер. Деякі сучасні блокбастери такі як «Король Лев» були зняті завдяки цьому інструменту.
- Timeline – інструмент що дозволяє створювати анімований контент а також взаємодіяти з аудіо файлами.
- Progressive Lightmapper – система для реалістичного освітлення предметів. Створює базові примітиви для предметів які необхідно освітлювати для подальшого полегшення роботи з освітленням.
- Autodesk Maya – інструмент для спрощення 3D анімацію на базі «накладання» скелету на персонажів.

Ігровий рушій з однаковим комфортом дозволяє створювати проект як у 2D так і у 3D стилі.

Завдяки продуманій оптимізації для роботи проектів на цьому рушії непотрібні значні ресурси системи.

Основною мовою для написання скриптів є C#.

На офіційному сайті рушія можна знайти детальну документацію всіх можливостей Unity. Також існую Unity Asset Store – магазин в якому художники можуть виставляти безкоштовні та платні ресурс паки.

Поширюється двигун у трьох варіантах:

- Personal – безкоштовна;
- Plus – 35 доларів за місяць
- Pro – 125 доларів за місяць

У доступних версіях немає різниці у можливостях для розробки і відрізняються лише максимальним можливим прибутком розробників.

Основним критерієм у виборі рушія є його багатоплатформовість, зручний інтерфейс, безкоштовна ліцензія для розробників що не отримують прибуток від проекту та можливість створення 2D застосунків. Саме тому обрано ігровий рушій Unity.

2.3 Microsoft Visual Studio

Для написання додаткових скриптів необхідно обрати комфортну IDE (Integrated Development Environment) – інтегроване середовище розробки для мови C# оскільки вона являється основною мовою розробки на ігровому рушії Unity.

Рекомендованим IDE для мови C# є Microsoft Visual Studio – офіційний додаток від компанії Microsoft – розробника мови C#. В ньому зручний інтерфейс (рис. 2.6) та він дозволяє додавати додатковий функціонал для рушія Unity.

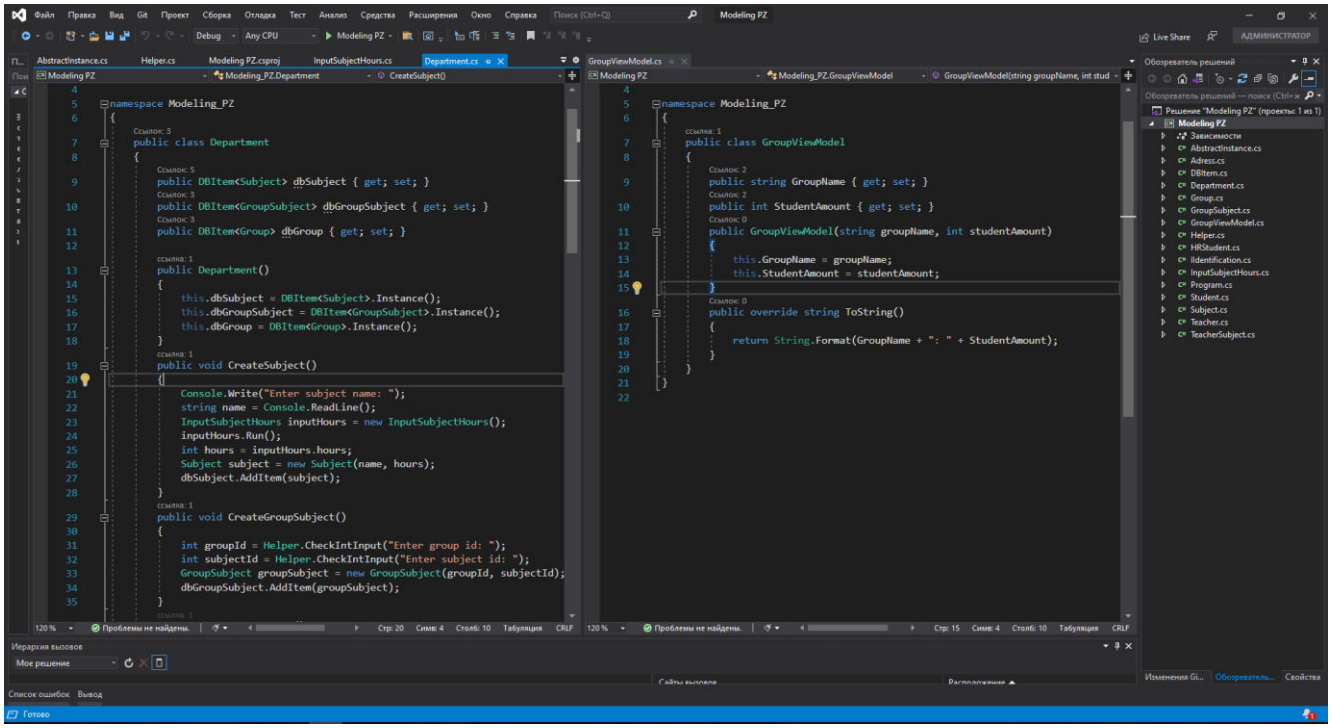


Рис. 2.6 – Інтерфейс Microsoft Visual Studio

Актуальною версією Visual Studio є додаток 2019 року. В цій редакції програми можна комфортно рефакторити програмний код що є ключовим при розробці ігор.

2.4 Загальний алгоритм розробки проекту

Розробка проекту відбуватиметься у 4 етапи:

- Проектування
- Розробка
- Тестування
- Реліз та подальша підтримка

На етапі проектування відбувається визначення мети гри та процес її розробки.

Мета гри це її жанр, ідея та середовище в якому будуть відбуватися дії в грі. Ідея гри – це те чим вона буде заволікати користувача та привертати його увагу. Те саме стосується і жанру.

Наступний крок у розробці гри це визначення її сеттингу – уявного світу де відбуваються події гри.

Наступним етапом є вибір платформи для якої буде гра, ігрового рушія та мови програмування. Не пропускаючи всі ці етапи можна уникнути в подальшому проблем на етапі розробки проекту.

Наступним крок це розробка проекту.

Зазвичай все починається з створення ігрового світу та персонажів. Потім поступово починають додавати рівні та урізноманітнювати гру.

Після створення першого рівня – прототипу гри, починається перевірка чи відповідає продукт очікуванням і чи це саме те, що хотіли створити.

Наступним етапом стає закрите бета-тестування на якому виявляють більшість помилок розробників – від некоректної логіки ігрових персонажів до різноманітних багів.

Одразу після виправлення помилок які виявлені на закритому тестуванні починається відкрите тестування.

Останній крок в реалізації проекту – його випуск та подальша підтримка у робочому стані. Також для подальшого розвитку гри варто її оновлювати додаючи новий контент який буде заохочувати нових гравців.

Зробивши аналіз етап розробки та проектування гри можна дійти висновку, що розробка гри відбувається за класичним алгоритмом розробки програмного забезпечення.

2.5 Аналіз потенційної аудиторії та споживачів

Одна з найголовніших помилок при створенні гри – не провести аналіз потенційної аудиторії. Оскільки вірогідність того, що грати у гру в яку хоче грати розробник доволі мала. Потенційна аудиторія – це люди, що будуть зацікавлені проектом та будуть у нього грати.

Для того щоб грамотно провести аналіз цільової аудиторії необхідно провести аналіз аудиторії ігор на які схожий ваш проект.

Для гри, що у розробці цільовою аудиторією є люди віком від 10 до 45 років.

Люди цього віку є активними користувачами комп’ютеру та активно грають у ігри. Для того щоб гра була популярною необхідно знайти баланс складності і зменшити «поріг входження у гру» - те на скільки інтуїтивно можна розібратися у механіках гри та її жанрі.

Отже, проект має змогу залучити велику кількість користувачів оскільки схожі ігри такого самого жанру досить популярні.

2.6 Актуальність проекту

Сучасна ігрова індустрія досить різноманітна та має величезну кількість користувачів. Щороку кількість нових гравців збільшується і задля вдоволення попиту необхідно створювати нові проекти.

Основуючись на популярності схожих проектів можна зробити висновок що в перспективі проект матиме успіх. Жанр платформер має низький поріг входу в гру та не потребують особливих навичків від гравців. Також ігри цього жанру не мають особливих вимог до потужності пристроїв гравців. Варто згадати і про те

що ігри цього жанру не потребують багато часу для того щоб досягти успіху під час проходження гри.

2.7 Мета проекту

Основна мета – це створення гри у жанрі платформер з більшістю атрибутів цього жанру та деякими елементами інших жанрів. Надати готовий продукт потеційним гравцям та покращити навички володіння ігровим рушієм для роботи з наступними проектами.

2.8 Функціонал проекту

Проект що в розробці – це відеогра у жанрі 2D платформер, основною метою якої є надати можливість приемно провести вільний час.

Беручи це до уваги, гра має відповідати наступним критеріям:

- Зрозумілий ігровий процес
- Інтуїтивно зрозумілі механіки
- Зручний та приємний інтерфейс
- Предмети для збирання
- Перешкоди, що ускладнюють проходження рівнів
- Після завершення спроби меню переходу в головне меню чи пропонуватися нова спроба

Це є базовий функціонал для гри, тому має бути реалізован у саму першу чергу.

2.9 Моделювання об’єктів проектування

2.9.1 Діаграма прецедентів

Uml Use Case diagram (діаграма прецедентів) – діаграма, яка графічно показує взаємодію користувача з програмою. Вона показує різні типи використання, які має система.

Дана діаграма створюється за допомогою об'єктів таких як: Actor, Use case, Package.

Actor - представляє роль користувача, який взаємодіє з системою, яку ви моделюєте. Користувачем може бути людина, організація, машина або інша зовнішня система.(рис. 2.7)

Рис. 2.7 - Actor

Use case - цей тип діаграми UML має бути оглядом високого рівня взаємовідносин між акторами та системами, тому вона може бути чудовим інструментом для пояснення вашої системи нетехнічній аудиторії. (Рис. 2.8)

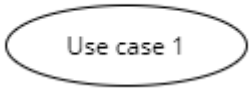


Рис. 2.8 - Use case

Package - використовуються, щоб показати залежності між різними пакетами в системі. Пакет, зображений у вигляді папки файлів, організовує елементи моделі, такі як варіанти використання або класи, у групи. (Рис. 2.9)

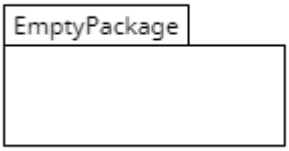


Рис. 2.9 - Package

Діаграма прецедентів для проекту.(рис. 2.10)

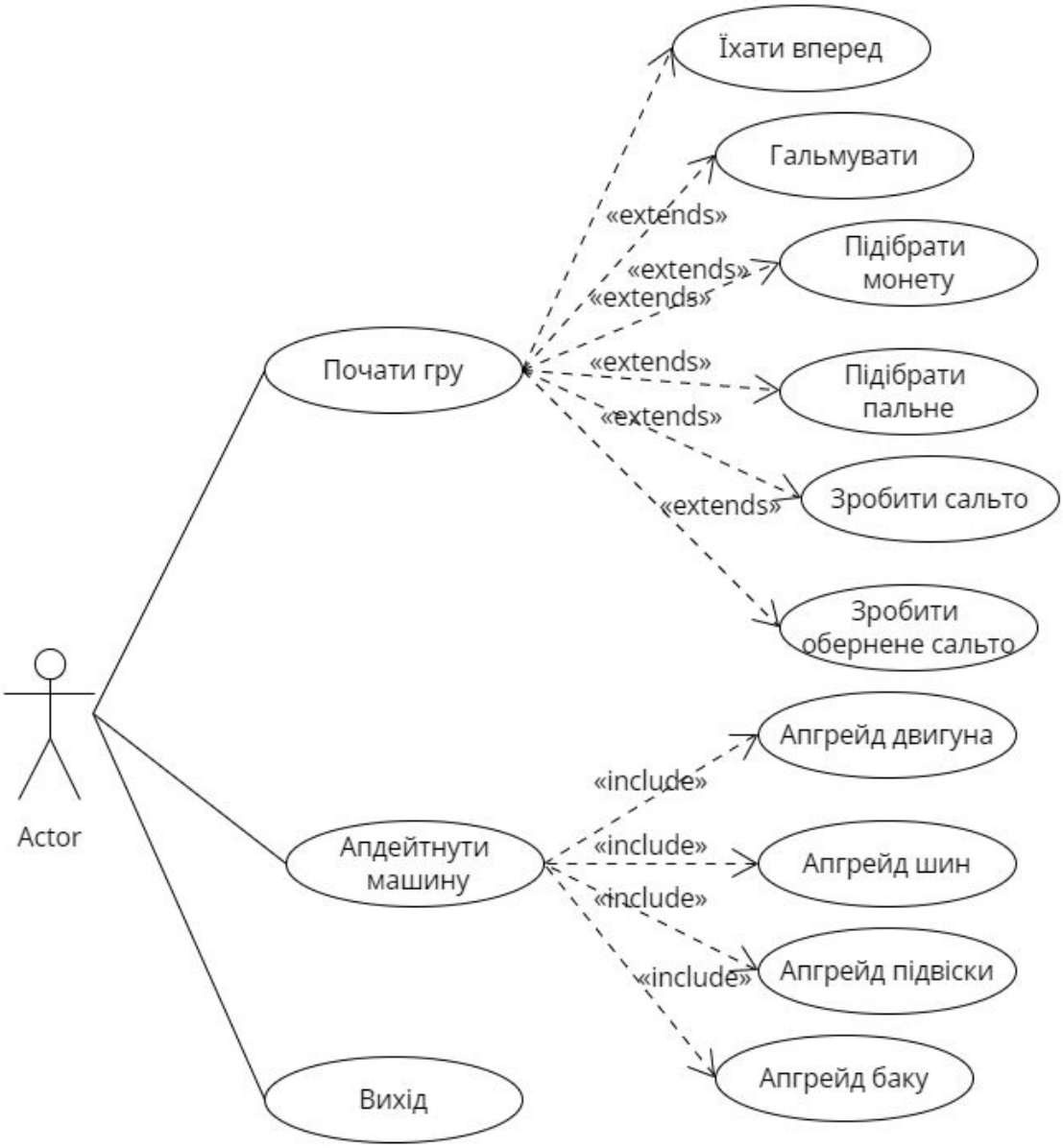


Рис. 2.10 - Діаграма прецедентів

2.9.2 Діаграма класів

Діаграма класів — це структурна діаграма мови моделювання UML, яка демонструє загальну структуру ієрархії класів системи, атрибутів, методів, інтерфейсів та взаємозв'язків між ними.

Діаграма класів для проекту. (Рис. 2.11)

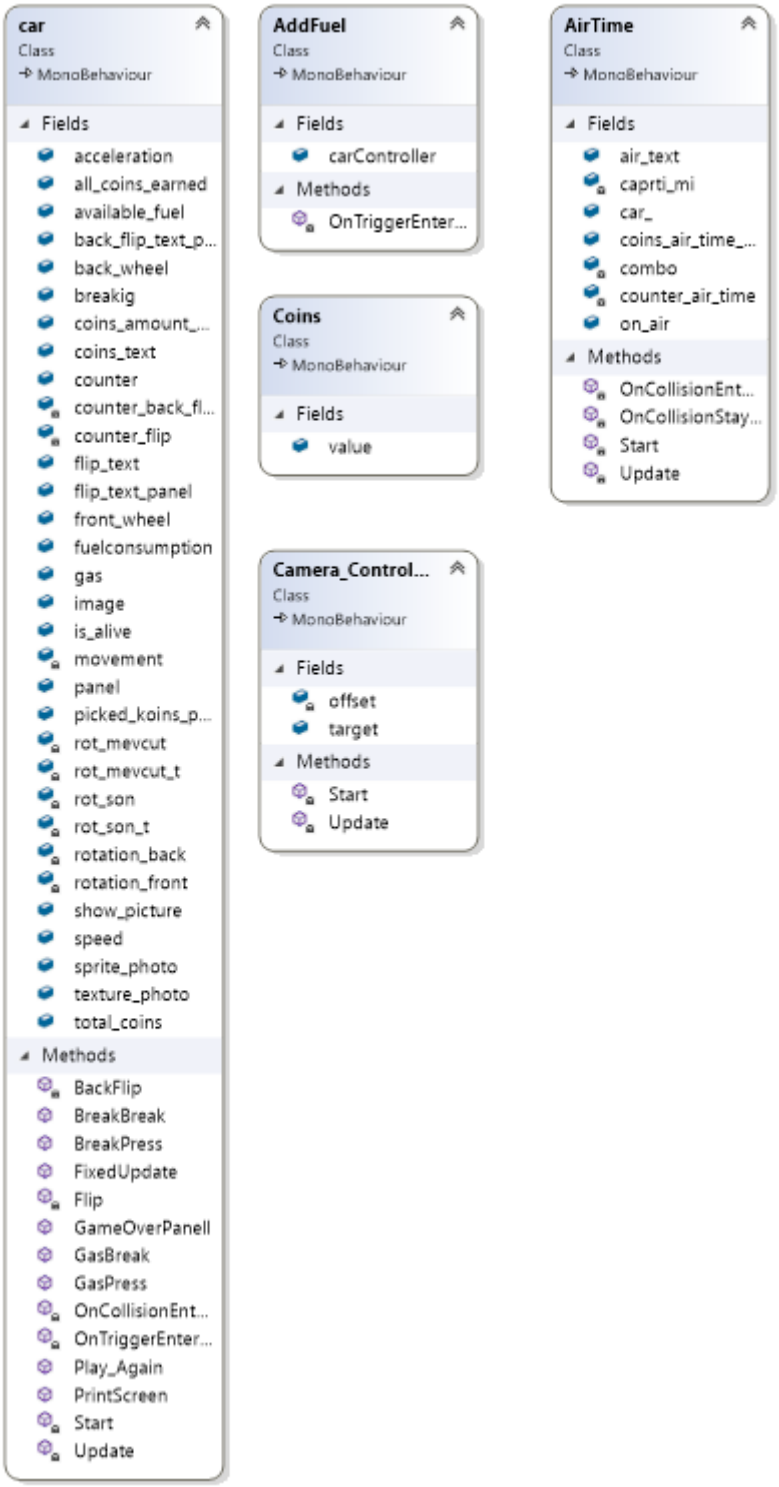


Рис. 2.11 - Діаграма класів

Висновок до розділу 2

У цьому розділі проведено детальний розбір того, що таке ігровий рушій, основні сучасні ігрові рушії – Unreal Engine, Cry Engine та Unity. Виділені переваги кожного рушія та особливості їх використання, зручність інтерфейсів для розробників, наявність документації. Для розробки обрано ігровий рушій Unity 2019, саме цьому для написання скриптів обрано IDE Microsoft Visual Studio – воно найбільш комфортне та доповнює функціонал ігрового рушія.

Наступним етапом було обрано алгоритм розробки проекту. Описано процес розробки гри. Проведено аналіз потенційної аудиторії та споживачів, визначено актуальність гри та мету проекту.

У завершення визначено базовий функціонал який необхідно розробити у першій версії гри.

3 РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1 Створення проекту у ігровому рушії Unity 2019

Реалізація гри починається з завантаження ігрового рушія Unity з офіційного сайту https://unity3d.com/get-unity/download. На цьому сайті завантажуємо та інсталюємо Unity Hub.

Наступним етапом є завантаження Editor Version.(рис. 3.1) Наразі актуальною версією є 2021.3.2f1 LST (LTS - Long Time Support).

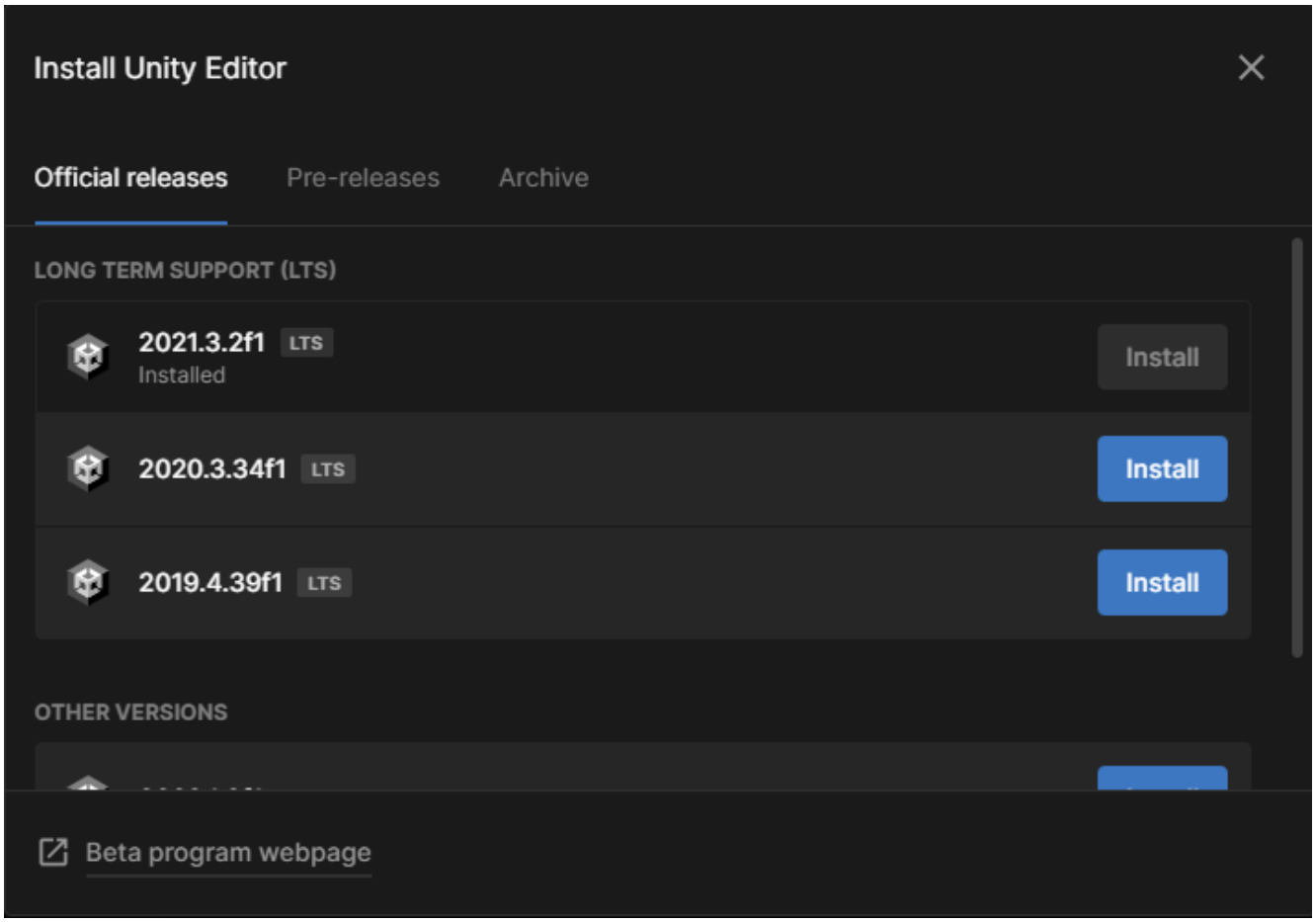


Рис. 3.1 - Unity Editor Version

Одразу після цього можна створювати новий проект.(рис. 3.2) Тут ми обираємо тип проекту серед перелічених який у нас буде – в нашому випадку це 2D та даємо йому назву не забувши обирати місце зберігання.

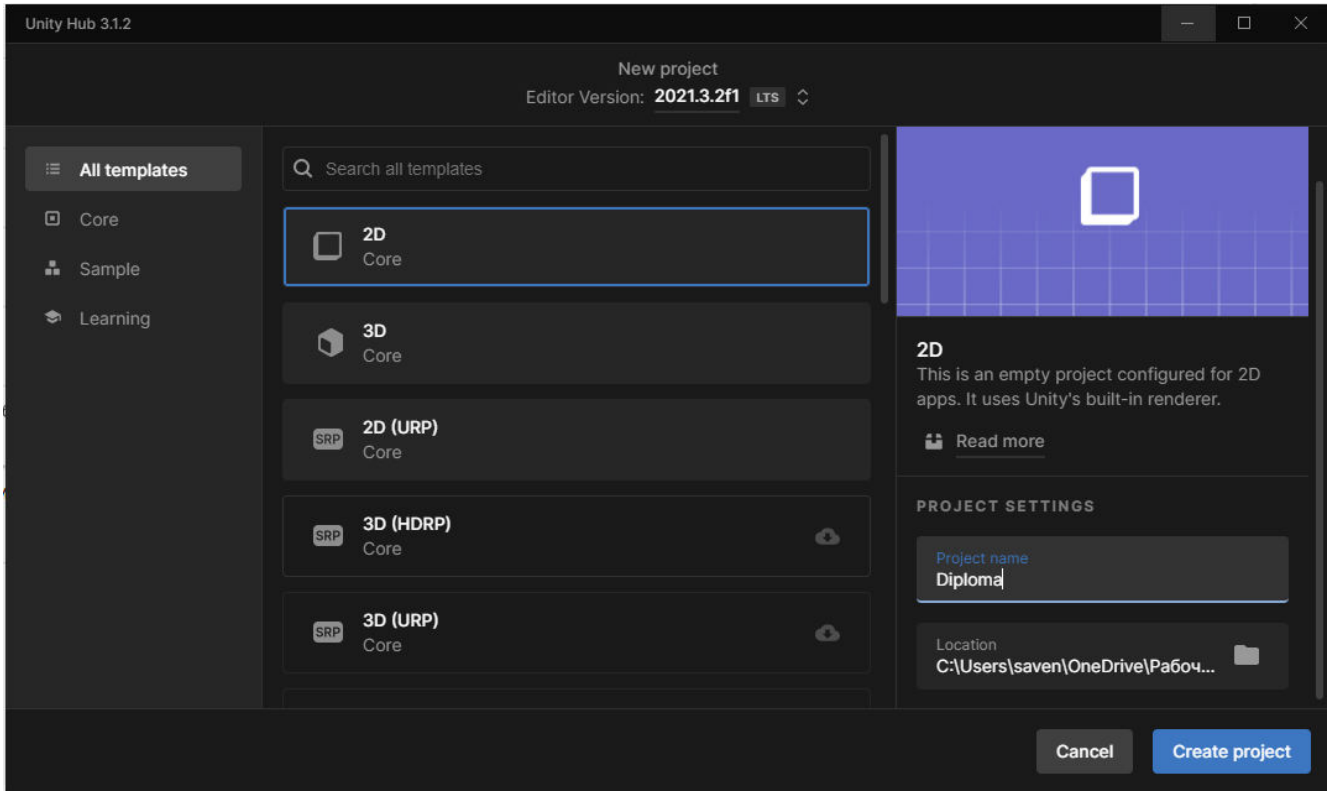


Рис. 3.2 – Меню створення нового проекту

Завершивши генерацію нового проекту відкривається вікно яке і є безпосередньо інструментом розробки рушія(рис. 3.3).

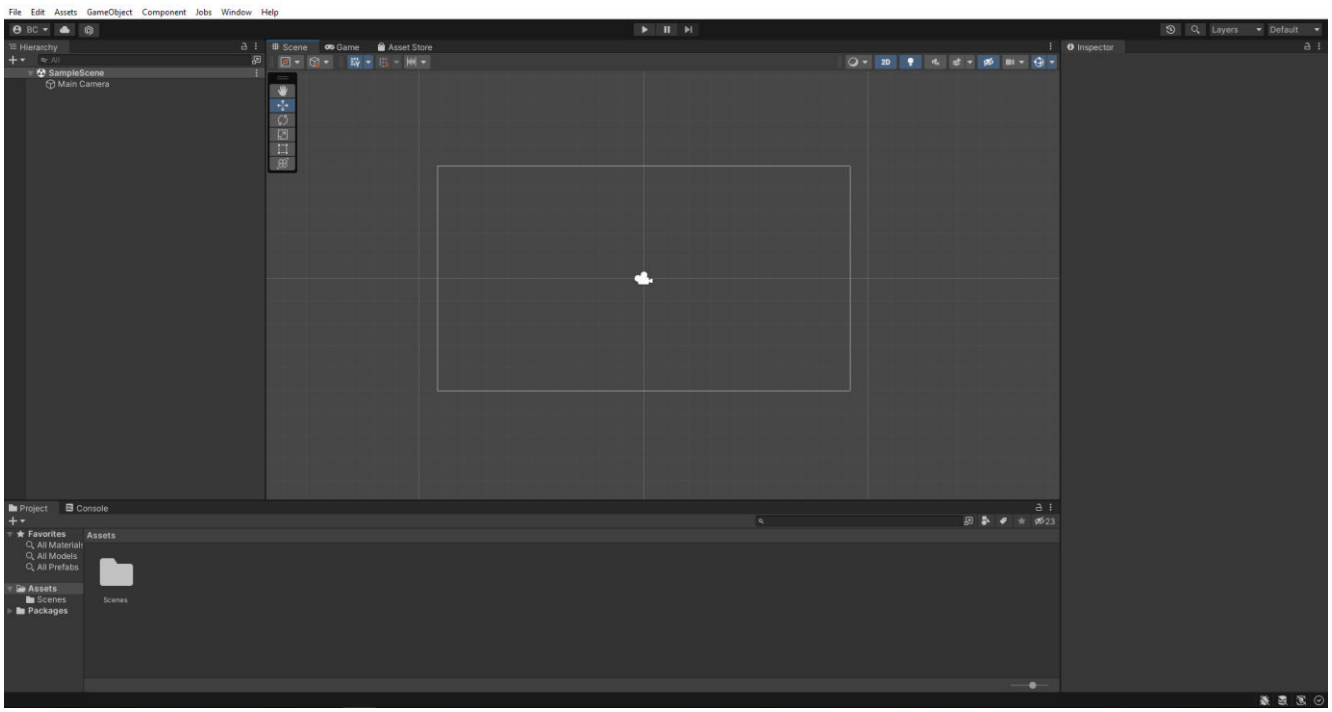


Рис. 3.3 – Вікно нового проекту

3.2 Графічне оформлення гри

Головною частиною будь-якої сучасної гри є її візуальна складова. Вона має бути приємною очам та не занадто строкатою.

Під час роботи з графічною складовою гри використовують спрайти (від англійського Sprite). Це одне або декілька зображень з яких роблять анімації руху об’єктів.

Для цього проекту були взяті безкоштовні спрайти з магазину ігрового рушія.

Для роботи зі спрайтами у ігровому рушії Unity є спеціальний інструмент Sprite Editor. (рис. 3.4)

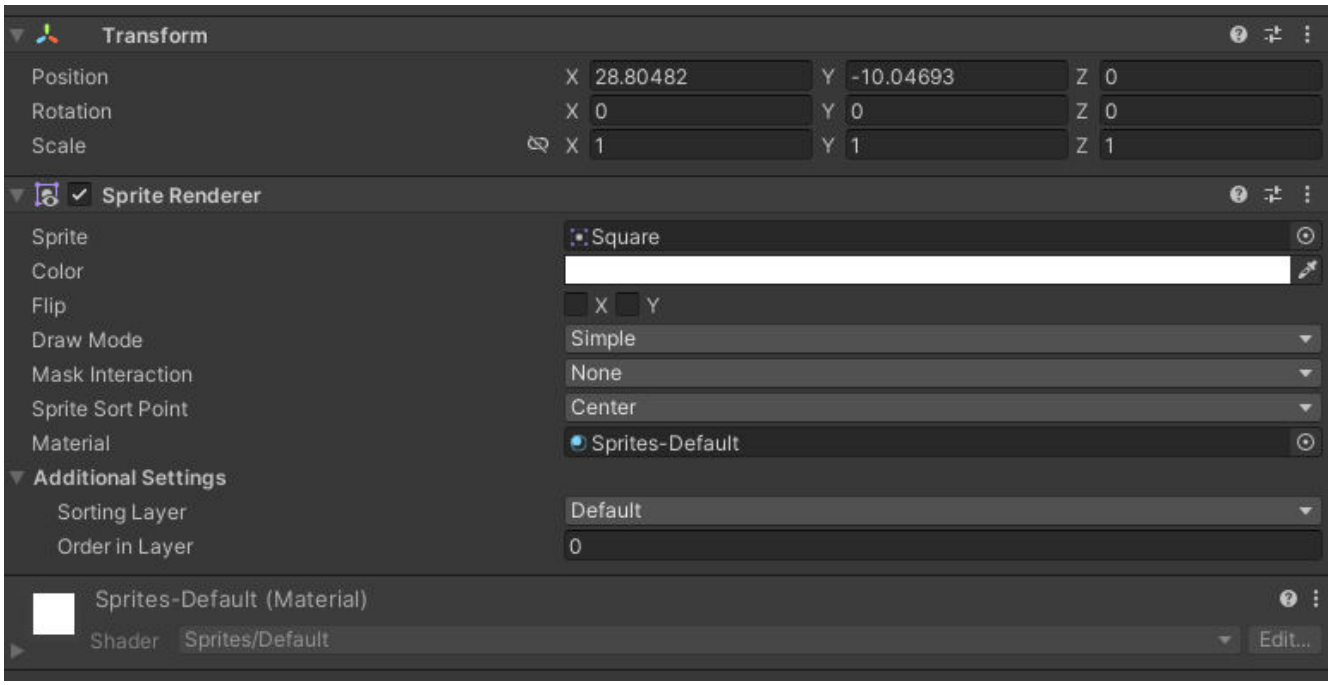


Рис. 3.4 - Sprite Editor

3.2.1 Спрайти машини, колес та водія

За основу взято спрайт машини в яку додано спрайт водія та колеса. (рис. 3.5)

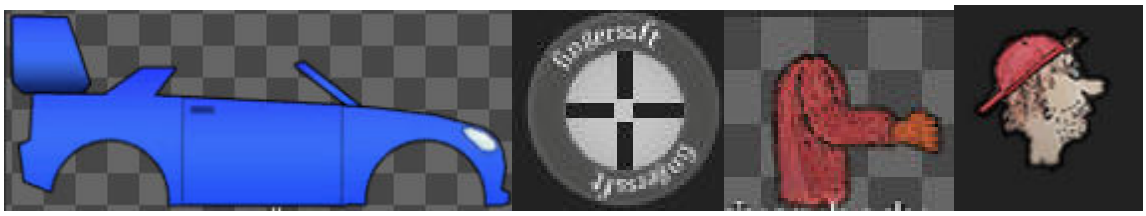


Рис. 3.5 – Спрайти машини та водія

3.2.2 Спрайти навколишнього середовища

Автомобілю необхідно рухатись по поверхі чогось і оскільки за основу взято спрайт автомобіля схожого на ралійний то вирішено в якості поверхні для їзди використати спрайт землі з травою.(рис. 3.6)

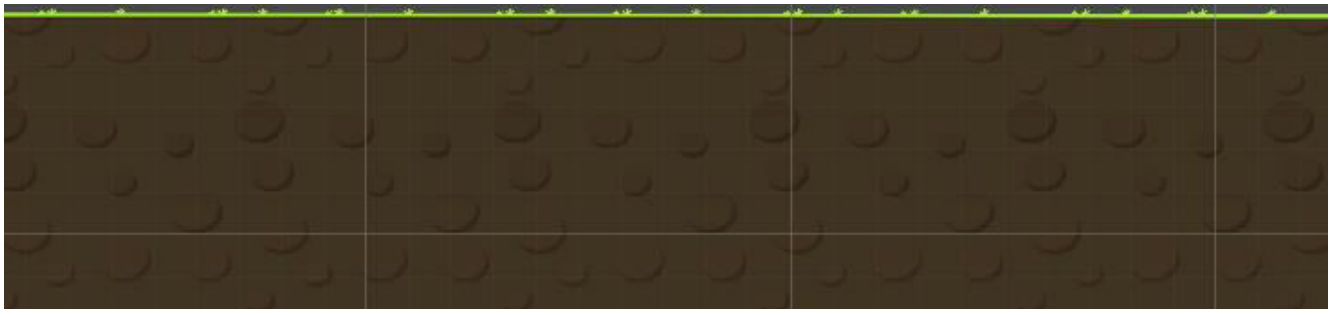


Рис. 3.6 – Спрайт землі

У якості спрайтів для фонового зображення обрано спрайт блакитно-рожевого неба (рис. 3.7) та спрайти хмар з деревами(рис. 3.8).

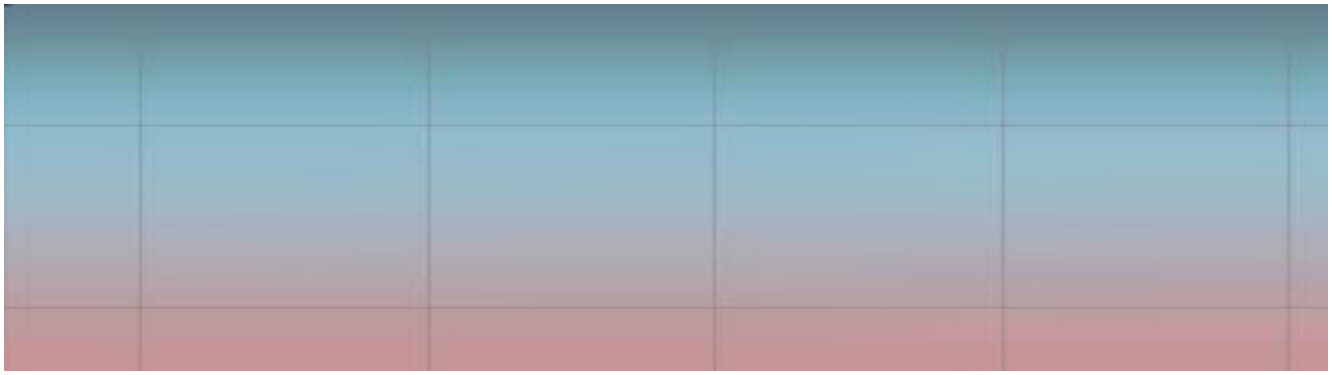


Рис. 3.7 – Спрайт заднього фону

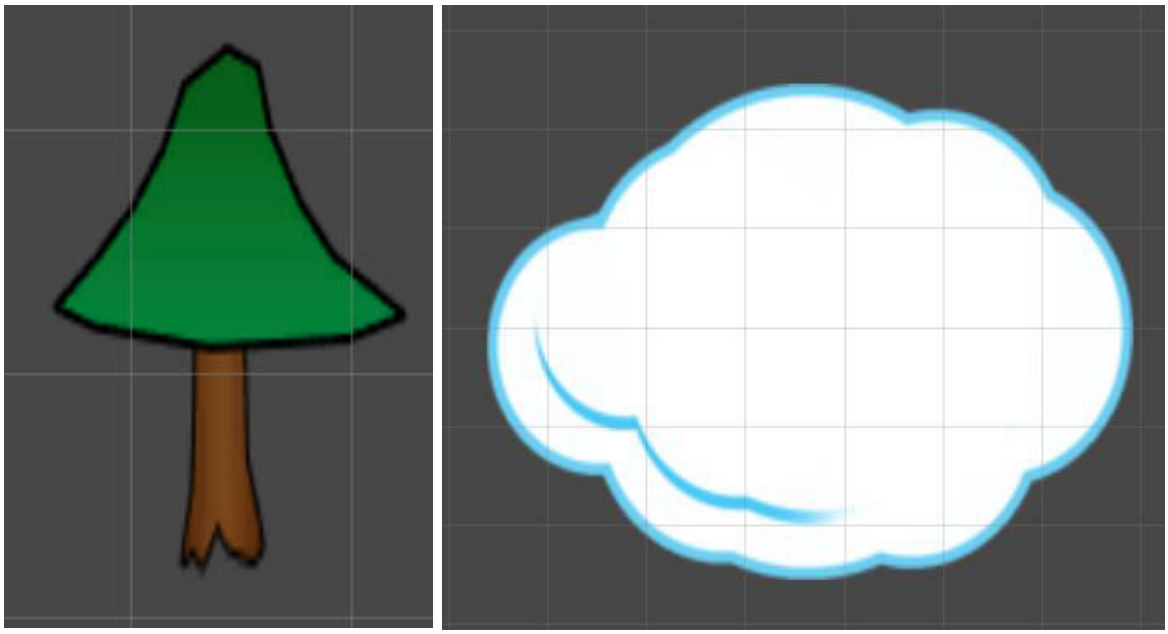


Рис. 3.8 – Спрайти хмар з деревами

3.2.3 Спрайти збираємих предметів

Для руху автомобіля необхідне пальне – наш випадок не виключення. Для того щоб поповнювати його запас під час руху рівнем на трассі будуть розташовані каністри. Також під час руху буде можливість збирати монети за які після завершення проходження можна буде покращити характеристики свого авто.(рис. 3.9)

Рис. 3.9 – Спрайти монет та каністри

3.3 Реалізація руху та підбору предметів.

Перш за все необхідно створити сцену на якій будуть відбуватися всі події. Після цього на сцену додаються спрайти. Для взаємодії всіх предметів вони повинні мати атрибут RigidBody 2D (2D оскільки гра двовимірна)(рис. 3.10). Цей атрибут надає предметам властивостей твердого тіла.

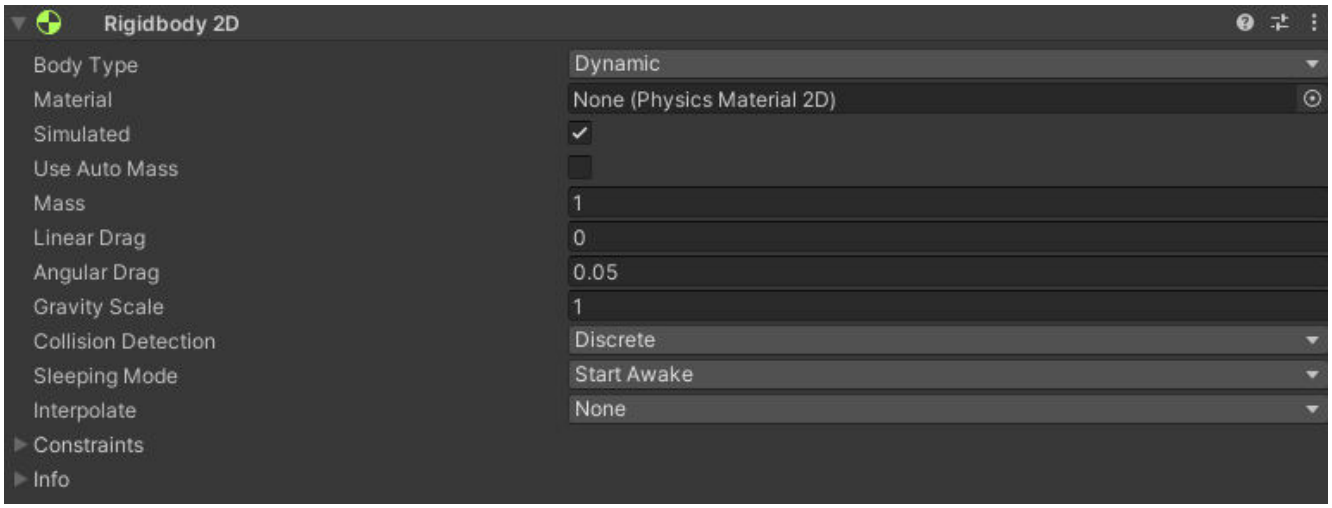


Рис. 3.10 – атрибут RigidBody 2D

Також предмети повинні мати колайдери (Collider). Вони необхідні для того щоб предмети не застрягали один в одному та взаємодіяли. Існують різні варіанти колайдерів – кругові, квадратні, полігональні та інші. Наприклад, для того щоб чітко позначити розміри спрайту автомобіля необхідно використовувати полігональний колайдер.(рис. 3.11) В цей самий час колесу вистачить і звичайного кругового колайдеру оскільки воно не має складних геометричних форм.

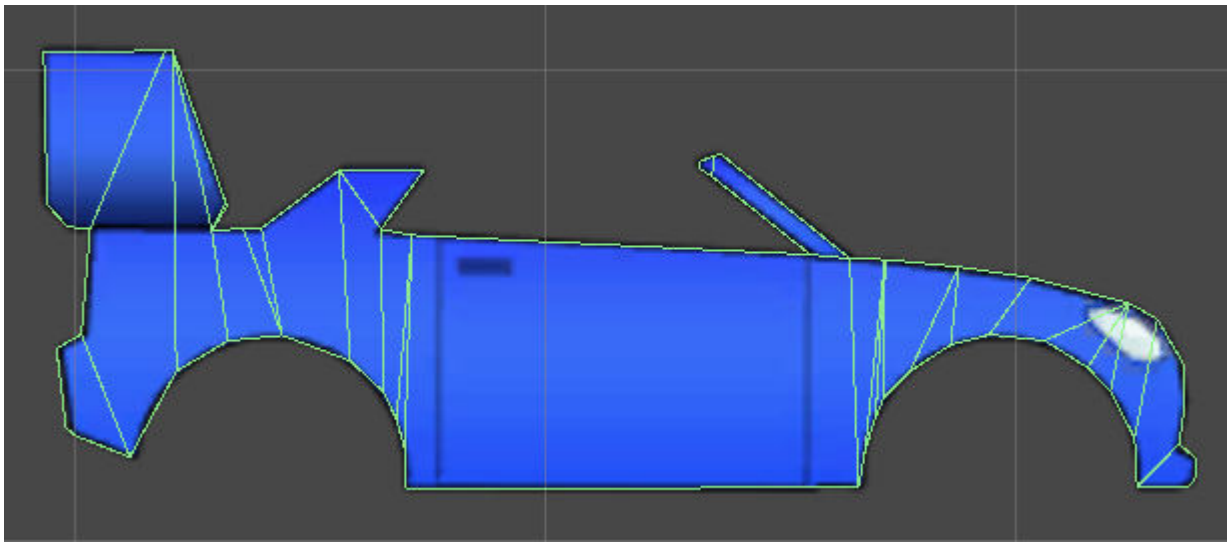


Рис. 3.11 – Приклад полігонального колайдеру

У ігровому рушії Unity є вбудований компонент Wheel Joint 2D,(рис. 3.12) який використовується для симуляції обертання колеса. Одними з параметрів є прискорення з яким відбувається обертання колеса та коефіцієнт його демпферування. Також він імітує кріплення колеса до транспорту шляхом пружини і дозволяє зберігати дистанцію до основної текстури на нерівностях поверхні під час руху.

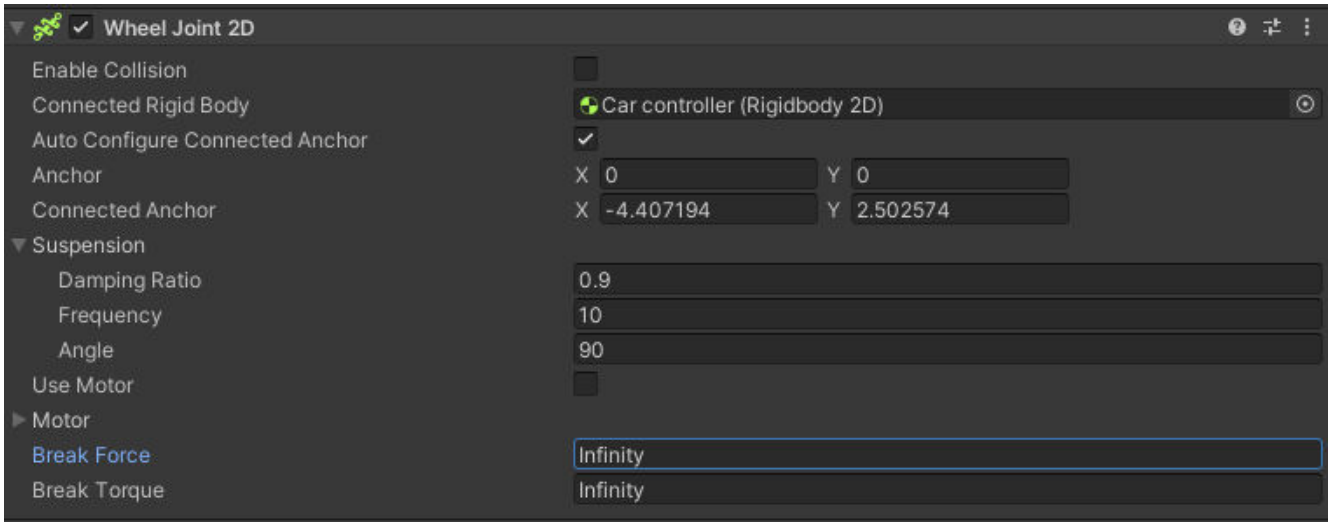


Рис. 3.12 - компонент Wheel Joint 2D

Однак вбудований інструмент дозволяє лише симулювати обертання. Для вказання напрямку руху написано наступний скрипт, який виконується у методі FixedUpdate, ключовою особливістю якого є те що він виконується кожний певний проміжок часу а не через якусь дію гравця.

```
public float speed;
private float movement;
public bool gas;
public bool breakig;
public float acceleration;
void Update()
{
    if (gas)
    {
        acceleration += 0.009f;
        if (acceleration > 1f)
        {
            acceleration = 1f;
        }
    }
    if (breakig)
    {
        acceleration -= 0.009f;
        if(acceleration < -1f)
        {
            acceleration = -1f;
        }
    }
    if (gas == false && breakig == false)
    {
        acceleration = 0;
        movement = speed * acceleration;
    }
}
```

Для реалізації використання автомобілем пального немає заготовлених скриптів тож його треба написати також власноруч. Також у цей скрипт входить і зміна зображення кількості пального для інтерфейсу гравця.(рис. 3.13)

```
public float available_fuel = 1;
public float fuelconsumption = 0.0000002f;
public Image image;
```

```
public void Update()
{
    image.fillAmount = available_fuel;
}
public void FixedUpdate()
{
    available_fuel -= fuelconsumption  * Time.fixedDeltaTime;
}
```

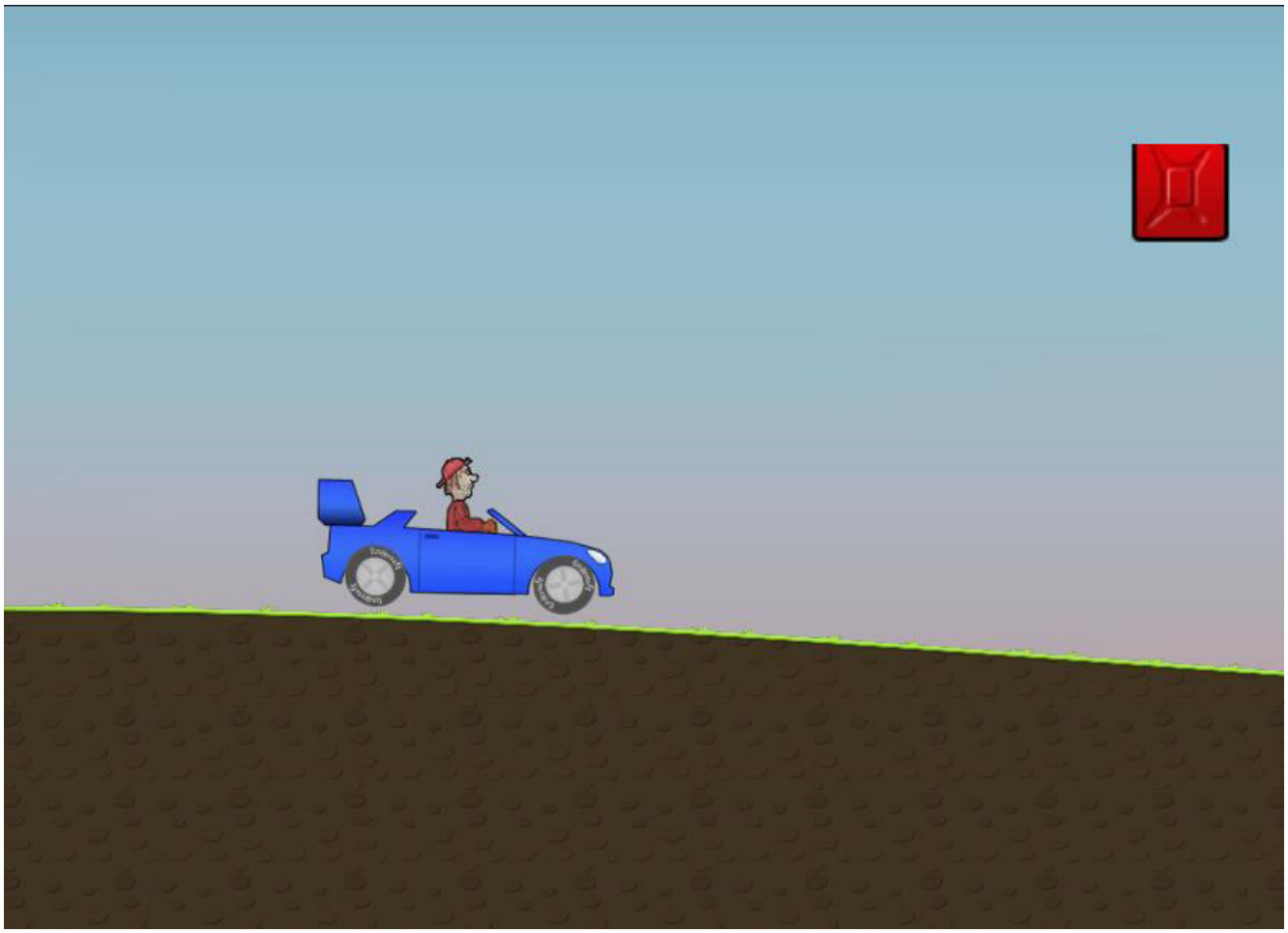


Рис. 3.13 – Інтерфейс пального

Однак крім використання пального потрібно ще й реалізувати механізм збирання його впродовж руху рівнем. Для цього створюється спрайт каністри(рис. 3.14) та скрипт що додає пальне. Додавши до спрайту колайдер він стає повноцінним об’єктом що взаємодіє з іншими на сцені.

Рис. 3.14 – Спрайт каністри

```
public car carController;
private void OnTriggerEnter2D(Collider2D collision)
{
    carController.available_fuel = 1;
    Destroy(gameObject);
}
```

Варто також не забувати знищувати спрайт каністри після підбирання пального. За це відповідає метод Destroy().

Для вдосконалення авто під час гри необхідно збирати монети. Вони мають різний номінал однак скрипт який взаємодіє з ними один для всіх. Для розуміння кількості зібраних монет під час проходження рівня є спеціальний елемент інтерфейсу. (рис. 3.15) Як і у випадку з пальним, об’єкти знищуються після їх збирання. Наступний код відповідає за збирання та збереження монет.

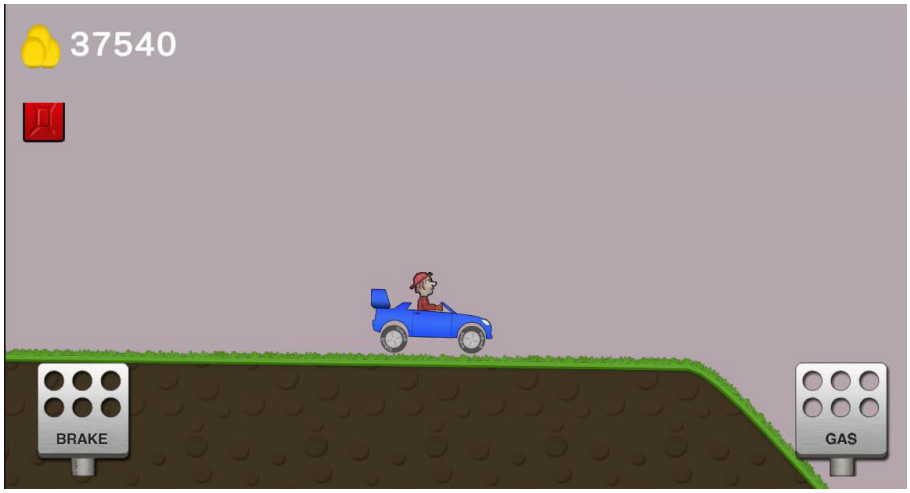


Рис. 3.15 – Елемент інтерфейсу

```
public int total_coins;
public TMP_Text coins_text;
public int all_coins_earned;

void Start()
{
     all_coins_earned = PlayerPrefs.GetInt("Altin");
}
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.tag == "altin")
    {
        int value = collision.GetComponent<Coins>().value;
        all_coins_earned += value;
        GameObject.Destroy(collision.gameObject);
        coins_text.text = total_coins.ToString();
    }
}
```

3.4 Реалізація бонусів за час в повітрі та сальто

За виконання гравцем трюків також є винагорода. Коли гравець виконує стрибок і знаходиться в повітрі йому нараховуються кошти.

Наступний код відповідає за нарахування коштів за проведений час в повітрі.

```
    float counter_air_time = 0;
    float combo = 0;
    public bool on_air = false;
    public GameObject car_;
    bool caprti_mi = false;
    public TMP_Text air_text;
    public int coins_air_time_earned;
    void Update()
    {
        if (counter_air_time > 0.5f)
        {
            combo++;
            counter_air_time = 0;
            air_text.text = "Air Time bonus: " + coins_air_time_earned * combo;
            car_.GetComponent<car>().all_coins_earned += coins_air_time_earned;
        }
            counter_air_time += Time.deltaTime;
```

```
    }
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.tag == "Player" || collision.gameObject.tag ==
"Teker(wheel)")
        {
            combo = 0;
            counter_air_time = 0;
            on_air = false;
            air_text.text = "";
        }
        if (collision.collider.tag == "kafa(head)")
        {
            car_.GetComponent<car>().PrintScreen();
            caprti_mi = true;
            car_.GetComponent<car>().is_alive = false;
        }
    }
    private void OnCollisionStay2D(Collision2D collision)
    {
        if (collision.gameObject.tag == "Player" || collision.gameObject.tag ==
"wheel")
        {
            combo = 0;
            counter_air_time = 0;
            on_air = false;
            air_text.text = "";
        }
    }
```

Також одним з його елементів є зображення в інтерфейсі користувача розмір отриманого бонусу за проведений час в повітрі(рис. 3.16).

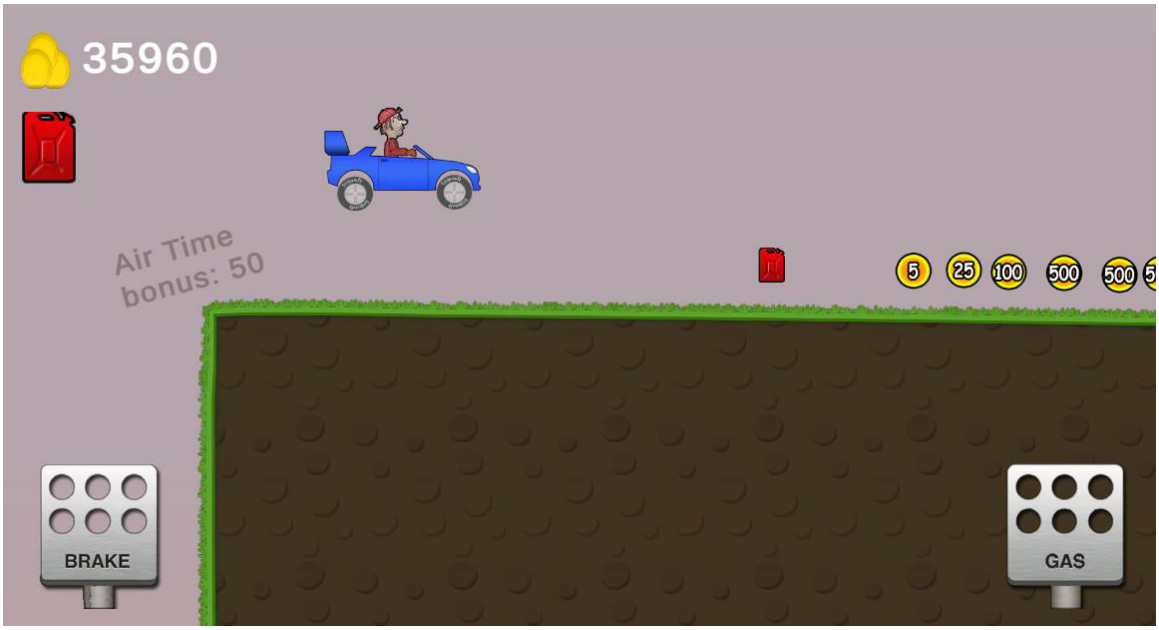


Рис. 3.16 – Відображення бонусу за час в повітрі

При виконанні гравцем трюку сальто та обернене сальто також нараховуються кошти.

Для реалізації цих трюків та нарахування за них коштів написано наступний код.

```
private float rotation_front;
private float rot_mevcut;
private float rot_son;
void Flip()
{
    rot_mevcut = transform.rotation.eulerAngles.z;
    if (rot_son < rot_mevcut)
    {
        rotation_front = rot_mevcut;
    }
    else if (rot_son > rot_mevcut && rot_son - rot_mevcut > 100)
    {
        rotation_front = rot_mevcut;
    }
    if (rotation_front - rot_mevcut > 300)
    {
        rotation_front = rot_mevcut;
        flip_text.text = "Flip + " + coins_amount_value;
        all_coins_earned += coins_amount_value;
        counter_flip++;
    }
    rot_son = rot_mevcut;
}
```

3.5 Інтерфейс завершення рівня

Гра завершується у випадку коли у гравця закінчується пальне або авто перевертається. Для цього написано наступний скрипт.

```
public bool is_alive = true;
void Update()
{
    if (!is_alive)
    {
        return;
    }
    if (available_fuel > 0)
    {
        Movement()
    }
    else
    {
        is_alive=false;
    }
}
private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.collider.tag == "kafa(head)")
    {
        is_alive = false;
    }
}
```

Після завершення гри гравцю відкривається меню(рис. 3.17)

Рис. 3.17 – Ігрове меню після програшу

В цьому меню показано скільки було зроблено трюків та скільки зароблено монет. Також показано сам момент коли гравець закінчив гру через переветання авто чи закінчення пального. За створення такого зображення відповідає наступний код.

```
    public Sprite sprite_photo;
    public Texture2D texture_photo;

    public void PrintScreen()
    {
        if(!is_alive)
        {
            return;
        }
        all_coins_earned += total_coins;
        PlayerPrefs.SetInt("Altin", all_coins_earned);
        Texture2D text = new Texture2D(Screen.width / 2, Screen.height / 2,
TextureFormat.RGB24, false);
        texture_photo = new Texture2D(Screen.width / 2, Screen.height / 2);
        text.ReadPixels(new Rect(Screen.width / 4, Screen.height / 4, Screen.width /
2, Screen.height / 2), 0, 0);
        text.Apply();
        texture_photo = text;
```

```
        text.Compress(false);
        sprite_photo = Sprite.Create(texture_photo, new Rect(0, 0, text.width,
text.height), new Vector2(0, 0));
        is_alive = false;
        GameOverPanell();
    }
```

Також цей код в подальшому викликає метод GameOverPanell(), який зображує гравцю статистику за проходження рівня. Наступний код створює цю панель.

```
public GameObject panel;
    public Image show_picture;
    public TMP_Text flip_text_panel;
    public TMP_Text back_flip_text_panel;
    public TMP_Text picked_koins_panel;
    private int counter_flip = 0;
    private int counter_back_flip = 0;

    public void GameOverPanell()
    {
        panel.SetActive(true);
        show_picture.sprite = sprite_photo;
        back_flip_text_panel.text = "Back Flip: " + counter_back_flip;
        flip_text_panel.text = "Front Flip: " + counter_flip;
        picked_koins_panel.text = "Coins: " + all_coins_earned;
    }
```

ВИСНОВКИ

У першому розділі проведено аналіз ігор жанру Платформер, визначено переваги та недоліки цього жанру. Розглянуто жанри ігор в цілому та перші ішри жанру Платформер.

У другому розділі більш детально розглянуто жанрову особливість ігор цього жанру. Проведено аналіз ігрових рушіїв, виявлено їх переваги та недоліки, обрано ігровий рушій для створення гри. Наступним кроком стало визначення етапів розробки гри, визначено цільову аудиторію, описано мету гри та її актуальність. Створено діаграми прецедентів та класів.

У третьому розділі відбувається реалізація гри. Визначено графічні складові проекту. Описано процес розробки гри. Продемонстровано ігрові механіки та інтерфейс та описали роботу головних методів і класів у грі.

Після виконання всіх етапів, реалізовано гру жанру Платформер, яка відповідає поставленим завданням. В подальшому проект можна розвивати додаючи нові механіки, мапи, авто тощо.

Список використаних джерел

1. Комп'ютерна гра [Електронний ресурс] – Режим доступу до ресурсу: https://esu.com.ua/search_articles.php?id=4393

2. Класифікація ігор за жанрами [Електронний ресурс] – Режим доступу до ресурсу: http://zps19.at.ua/publ/mediateksti_vid_ekspertiv/klasifikacija_i_zhanri_komp_juternikh_i_videoigor/15-1-0-226.

3.Unreal Engine [Електронний ресурс] – Режим доступу до ресурсу: https://www.unrealengine.com/.

4.Cry Engine [Електронний ресурс] – Режим доступу до ресурсу: https://www.cryengine.com/.

5. Unity Game Engine [Електронний ресурс] – Режим доступу до ресурсу: https://unity.com/ru.

6. Unity Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unity3d.com/Manual/index.html.

7. Створення 2D на Unity [Електронний ресурс] – Режим доступу до ресурсу: https://gamedevacademy.org/how-to-build-a-complete-2d-platformer-in-unity/.

8. Пояснення про спрайти [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unity3d.com/ScriptReference/Sprite.html.

9. Гра Super Mario Bros [Електронний ресурс] – Режим доступу до ресурсу: https://mario.nintendo.com/

10. Гра Hill Climb Racing [Електронний ресурс] – Режим доступу до ресурсу: https://hillclimb-racing.com/

11. Гра Earn to Die [Електронний ресурс] – Режим доступу до ресурсу: https://earntodie.co/

12. Гра Max Fury - Road Warrior Racing [Електронний ресурс] – Режим доступу до ресурсу: https://www.smokoko.com/

13. Hocking J. Unity in Action. Multiplatform game development in C# with Unity 5 / Joseph Hocking., 2015. – 352 с.

14. Wolf M. J. P. Encyclopedia of Video Games: The Culture, Technology, and Art of Gaming / Ed. by Mark J. P. Wolf. Santa Barbara, – CA: Greenwood, 2012. – 140 c.
15. Herman H. Goldstine[en]. The Computer from Pascal to von Neumann. — Princeton University Press, 1980. - 125 c.
16. Kent, Steven (2010-06-16). The Ultimate History of Video Games: from Pong to Pokemon and beyond...the story behind the craze that touched our li ves and changed the world.
17. Crawford, Chris (1982). The Art of Computer Game Design.
18. Clearwater, David A. (2011). What Defines Video Game Genre? Thinking about Genre Study after the Great Divide. The Journal of the Canadian Game Studies Association.
19. Thorn A. Mastering Unity Scripting / Alan Thorn., 2015. – 380 c.
20. Донован Т. Грай! Історія видеоігор / Трістан Донован; пер. И. Вороніна. — М.: Біле Яблуко, 2014. – 45 с.

Додаток А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

РОЗРОБКА ГРИ В ЖАНРІ 2D ПЛАТФОРМЕР «HILL CLIMBING» З ВИКОРИСТАННЯМ ДВИГУНА UNITY

Виконав студент 4 курсу
групи ПД-44
Савенко Вадим Володимирович
Керівник роботи
PhD, Дібрівний Олесь Андрійович

Київ-2022

АНАЛОГИ

Назва	Переваги	Недоліки
Hill Climb Racing	Мультиплатформовість	Однотипність геймплею
	Різноманітність рівнів	
Earn to Die	Унікальне руйнування авто	Лише одна карта
		Не є безкоштовною
Gravity Defied: Trial Racing	Мінімальні вимоги потужності пристроїв	Лише для телефонів
Max Fury - Road Warrior Racing	Унікальний сетинг гри	Велика кількість внутрішніх транзакцій(Pay to win)

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – підвищення зацікавленості гравців в ігровому процесі ігор в жанрі платформер за рахунок кросплатформості додатку та зменшення вимог до ресурсів ігрового пристрою

Об'єкт дослідження – ігровий процес в жанрі платформер

Предмет дослідження – програмне забезпечення для реалізації гри жанру платформер

ТЕХНІЧНІ ЗАВДАННЯ

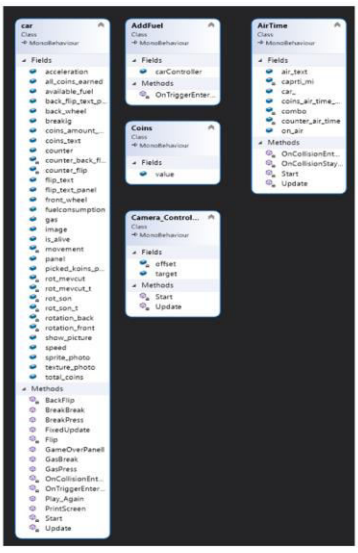
1.Створити головне ігрове меню
2.Додати меню покращення властивостей авто
3.Додати меню налаштувань
4.Розробити першу ігрову мапу
5.Розробити основні ігрові механіки
6.Реалізувати мультиплатформовість

4

ПРОГРАМНІ ТА ТЕХНІЧНІ ЗАСОБИ РЕАЛІЗАЦІЇ

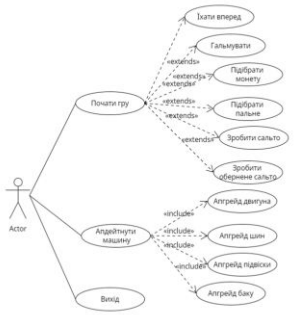
5

ДІАГРАМА КЛАСІВ



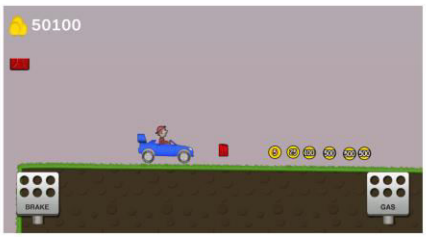
6

USE CASE ДІАГРАМА



7

ЕКРАННІ ФОРМИ



Основний ігровий геймплей

Меню після завершення рівня

8

ВИСНОВКИ

1.Проведено аналіз існуючих аналогів. Виявлено переваги і недоліки.
2.Проведено аналіз основних ігрових рушіїв.
3.Розроблено головне ігрове меню
4.Додано меню покращення властивостей авто
5.Створено меню налаштувань
6.Створено першу ігрову мапу
7.Розроблено основні ігрові механіки

9

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Савенко В.В., СУЧАСНІ ПРИНЦИПИ РОЗРОБКИ КАЗУАЛЬНИХ ІГОР, XIV НАУКОВО-ТЕХНІЧНА КОНФЕРЕНЦІЯ СТУДЕНТІВ ТА МОЛОДИХ ВЧЕНИХ «СУЧАСНІ ІНФОКОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ», ст. 96-97.

Савенко В.В., ФІЗИЧНИЙ РУШІЙ, ЯК НЕВІД'ЄМНА ЧАСТИНА МАЙБУТНЬОГО ВІДЕОІГОР, НАУКОВО-ТЕХНІЧНА КОНФЕРЕНЦІЯ ЗАСТОСУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ В ІНФОКОМУНІКАЦІЙНИХ ТЕХНОЛОГІЯХ, ст. 88-89.

10

ДЯКУЮ ЗА УВАГУ!

11

Додаток Б

```
ДОДАТОК Б
using UnityEngine;

public class AddFuel : MonoBehaviour
{
    public car carController;
    private void OnTriggerEnter2D(Collider2D collision)
    {
        carController.available_fuel = 1;
        Destroy(gameObject);
    }
}

using UnityEngine;
using TMPro;

public class AirTime : MonoBehaviour
{
    float counter_air_time = 0;
    float combo = 0;
    public bool on_air = false;
    public GameObject car_;
    bool caprti_mi = false;
    public TMP_Text air_text;
    public int coins_air_time_earned;
    // Start is called before the first frame update
    void Start()
    {
    }

    void Update()
    {
        if (counter_air_time > 0.5f)
        {
            combo++;
            counter_air_time = 0;
            air_text.text = "Air Time bonus: " + coins_air_time_earned * combo;
            car_.GetComponent<car>().all_coins_earned += coins_air_time_earned;
        }
            counter_air_time += Time.deltaTime;
    }
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.tag == "Player" || collision.gameObject.tag ==
"Teker(wheel)")
        {
            combo = 0;
            counter_air_time = 0;
```

```
            on_air = false;
            air_text.text = "";
        }
        if (collision.collider.tag == "kafa(head)")
        {
            car_.GetComponent<car>().PrintScreen();
            caprti_mi = true;
            car_.GetComponent<car>().is_alive = false;
        }
    }
    private void OnCollisionStay2D(Collision2D collision)
    {
        if (collision.gameObject.tag == "Player" || collision.gameObject.tag ==
"Teker(wheel)")
        {
            combo = 0;
            counter_air_time = 0;
            on_air = false;
            air_text.text = "";
        }
    }
}

using UnityEngine;

public class Camera_Controller : MonoBehaviour
{
    public Transform target;
    private Vector3 offset;
    // Start is called before the first frame update
    void Start()
    {
        offset = transform.position - target.position;
    }
    // Update is called once per frame
    void Update()
    {
        transform.position = target.position + offset;
    }
}

using UnityEngine;
using TMPro;
using UnityEngine.UI;

public class car : MonoBehaviour
{
    public WheelJoint2D back_wheel;
    public WheelJoint2D front_wheel;
    public float speed;
```

```
private float movement;
private float rotation_front;
private float rot_mevcut;
private float rot_son;
private float rotation_back;
private float rot_mevcut_t;
private float rot_son_t;
public int total_coins;
public TMP_Text coins_text;
public float available_fuel = 1;
public float fuelconsumption = 0.0000002f;
public Image image;
public bool is_alive = true;
public Sprite sprite_photo;
public Texture2D texture_photo;
public TMP_Text flip_text;
public int coins_amount_value;
public float counter = 0.1f;
public GameObject panel;
public Image show_picture;
public TMP_Text flip_text_panel;
public TMP_Text back_flip_text_panel;
public TMP_Text picked_koins_panel;
private int counter_flip = 0;
private int counter_back_flip = 0;
public bool gas;
public bool breakig;
public float acceleration;
public int all_coins_earned;

// Start is called before the first frame update
void Start()
{
    rot_mevcut = transform.rotation.eulerAngles.z;
    rotation_front = rot_mevcut;
    rot_son = rot_mevcut;
    rot_mevcut_t = transform.rotation.eulerAngles.z;
    rotation_back = rot_mevcut_t;
    rot_son_t = rot_mevcut_t;
    acceleration = 0;
    counter_flip = 0;
    counter_back_flip = 0;
    all_coins_earned = PlayerPrefs.GetInt("Altin");
}

// Update is called once per frame
void Update()
{
    if (!is_alive)
    {
```

```
            return;
        }
        if (available_fuel > 0)
        {
            if (gas)
            {
                acceleration += 0.009f;
                if (acceleration > 1f)
                {
                    acceleration = 1f;
                }
            }
            if (breakig)
            {
                acceleration -= 0.009f;
                if(acceleration < -1f)
                {
                    acceleration = -1f;
                }
            }
            if (gas == false && breakig == false)
            {
                acceleration = 0;
            }
            movement = speed * acceleration;
        }
        image.fillAmount = available_fuel;
        coins_text.text = all_coins_earned.ToString();
        if (available_fuel < 0)
        {
            is_alive = true;
            PrintScreen();
            GameOverPanell();
        }
        if (flip_text.text != "")
        {
            counter -= Time.deltaTime;
        }
        if (counter <= 0)
        {
            flip_text.text = "";
            counter = 0.1f;
        }
    }
    public void FixedUpdate()
    {
        if (movement == 0)
        {
            back_wheel.useMotor = false;
            front_wheel.useMotor = false;
```

```
        }
        else
        {
            back_wheel.useMotor = true;
            front_wheel.useMotor = true;
            JointMotor2D motore = new JointMotor2D();
            motore.motorSpeed = movement;
            motore.maxMotorTorque = 10000;
            back_wheel.motor = motore;
            front_wheel.motor = motore;
        }
        available_fuel -= fuelconsumption  * Time.fixedDeltaTime;
        Flip();
        BackFlip();
    }
    void Flip()
    {
        rot_mevcut = transform.rotation.eulerAngles.z;
        if (rot_son < rot_mevcut)
        {
            rotation_front = rot_mevcut;
        }
        else if (rot_son > rot_mevcut && rot_son - rot_mevcut > 100)
        {
            rotation_front = rot_mevcut;
        }
        if (rotation_front - rot_mevcut > 300)
        {
            rotation_front = rot_mevcut;
            flip_text.text = "Flip + " + coins_amount_value;
            all_coins_earned += coins_amount_value;
            counter_flip++;
        }
        rot_son = rot_mevcut;
    }
    void BackFlip()
    {
        rot_mevcut_t = transform.rotation.eulerAngles.z;
        if (rot_son_t > rot_mevcut_t)
        {
            rotation_back = rot_mevcut_t;
        }
        else if (rot_son_t < rot_mevcut_t && rot_mevcut_t - rot_son_t > 100)
        {
            rotation_back = rot_mevcut_t;
        }
        if (rot_mevcut_t - rotation_back > 300)
        {
            rotation_back = rot_mevcut_t;
            flip_text.text = "Back Flip + " + coins_amount_value;
```

```
            all_coins_earned += coins_amount_value;
            counter_back_flip++;
        }
        rot_son_t = rot_mevcut_t;
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.tag == "altin")
        {
            int value = collision.GetComponent<Coins>().value;
            all_coins_earned += value;
            GameObject.Destroy(collision.gameObject);
            coins_text.text = total_coins.ToString();
        }
    }
    public void PrintScreen()
    {
        if(!is_alive)
        {
            return;
        }
        all_coins_earned += total_coins;
        PlayerPrefs.SetInt("Altin", all_coins_earned);
        Texture2D text = new Texture2D(Screen.width / 2, Screen.height / 2,
TextureFormat.RGB24, false);
        texture_photo = new Texture2D(Screen.width / 2, Screen.height / 2);
        text.ReadPixels(new Rect(Screen.width / 4, Screen.height / 4, Screen.width /
2, Screen.height / 2), 0, 0);
        text.Apply();
        texture_photo = text;
        text.Compress(false);
        sprite_photo = Sprite.Create(texture_photo, new Rect(0, 0, text.width,
text.height), new Vector2(0, 0));
        is_alive = false;
        GameOverPanell();
    }
    private void OnCollisionEnter2D(Collision2D collision)
    {

        //head-ground
        if (collision.collider.tag == "kafa(head)")
        {
            is_alive = false;
        }
    }
    public void Play_Again()
    {
        Application.LoadLevel(Application.loadedLevel);
    }
    public void GameOverPanell()
```

```
    {
        panel.SetActive(true);
        show_picture.sprite = sprite_photo;
        back_flip_text_panel.text = "Back Flip: " + counter_back_flip;
        flip_text_panel.text = "Front Flip: " + counter_flip;
        picked_koins_panel.text = "Coins: " + all_coins_earned;
    }
    public void GasPress()
    {
        gas = true;
    }
    public void GasBreak()
    {
        gas = false;
    }
    public void BreakPress()
    {
        breakig = true;
    }
    public void BreakBreak()
    {
        breakig = false;
    }
}

using UnityEngine;

public class Coins : MonoBehaviour
{
    public int value;
}
```