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 БАГАТОКОРИСТУВАЦЬКОЇ ГРИ В ЖАНРІ АРКАДНОЇ ГОНКИ МОВОЮ C#»**

Виконав: студент 5 курсу, групи ППЗ–51 спеціальності
121 Інженерія програмного забезпечення.
(шифр і назва спеціальності)
Прежина А.А.
(прізвище та ініціали)
Керівник Шевченко С.М.
(прізвище та ініціали)
Рецензент_____
(прізвище та ініціали)

Київ – 2021

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
“____” _____2021 року

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

Прежина Анастасія Андріївна

1.Тема роботи: _Розробка багатокористувацької гри в жанрі аркадної гонки мовою C#_____
Керівник роботи_ Шевченко С.М._____,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від “12” березня_2021 р. №65.
2. Строк подання студентом роботи _“ 01 ” червня_2021 р._____
3. Вхідні дані до роботи: мова програмування C#, науково-технічна література на тему розробки відеоігор.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).

4.1. Дослідження поняття відеогри.

4.2. Основні підходи при розробці відеоігор.

4.3. Дослідження основних проблем при розробці відеоігор та методів їх вирішенню.

4.4. Проєктування і розробка продукту.

5. Перелік графічного матеріалу.

5.1. Приклади інших продуктів на ринку

5.2. Порівняння інструментів для розробки.

5.3. Діаграма класів програми.

5.4. Компоненти програми.

6. Дата видачі завдання__19.04.2021 р._____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	19.04.2021	Виконано
2	Дослідження на тему розробки ігор	26.04.2021	Виконано
3	Розробка архітектури програми	03.05.2021	Виконано
4	Створення на багато користувачів аспекту	10.05.2021	Виконано
5	Створення персонажа і його поведінки	14.05.2021	Виконано
6	Реалізація загальної логіки гри	18.05.2021	Виконано
7	Створення інтерфейсу користувача	20.05.2021	Виконано
8	Вступ, Висновки, реферат	22.05.2021	Виконано
9	Створення демонстраційних матеріалів	23.05.2021	Виконано
10	Попередній захист роботи	25.05.2021	
11	Подання роботи в деканат	01.06.2021	

Студент_____ Прежина А.А.

Керівник роботи _____ Шевченко С.М.

РЕФЕРАТ

Текстова частина бакалаврської роботи: 37 с., 4 табл., 22 рис., 20 джерел.

РОЗРОБКА ІГОР, ПРОЦЕДУРНА ГЕНЕРАЦІЯ, БАГАТОКОРИСТУВАЦЬКА ГРА, UNITY, PHOTON UNITY NETWORKING

Відеоігри вже давно стали повсякденною частиною нашого життя, від звичайного дозвілля і навчання до реабілітації дітей з ДЦП. Актуальність дослідження підтверджується також умовами, що виникли на даному етапі у світі в зв'язку з пандемією коронавірусу. Бо, коли на міжнародній арені відбувається спад у різних галузях бізнесу, саме «зростання замовлень відчули розробники ігор — у низці компаній справи пішли навіть краще, ніж до карантину. Кількість вакансій в геймдев індустрії у квітні зросла на 11%» [1].

Об`єкт дослідження – розробка відеоігор.

Предмет дослідження – багатокористувацькі ігри.

Мета роботи – здійснення аналізу сучасних інструментальних засобів для побудови відеоігор.

Наукова розробка присвячена створенню багатокористувацької відеогри в жанрі аркадних гонок з використанням ігрового двигуна Unity. Даний багатоплатформний ігровий двигун був обраний через простоту і універсальність в розробці в порівнянні з конкурентами

ЗМІСТ

стор.

ВСТУП

Для багатьох із нас відеоігри давно стали повсякденною частиною нашого життя як новий спосіб задоволення соціальних і емоційних потреб. Суть відеоігор у тому, щоб передати той або інший досвід і разом з ним емоційний відгук.

Відеоігри самі по собі дуже різноманітні, вони різняться жанрами, стилями, підходами розповісти ту чи іншу історію, за це дехто вважає їх окремою галуззю мистецтва чи свого роду інтерактивним фільмом. Вони затягують нас, переносять в незвичайні та незнайомі умови, змушує активно взаємодіяти з середовищем, вирішувати завдання. Наприклад, люди грають у різного роду симулятори, щоб відчути себе іншою людиною або істотою, можливо спробувати іншу професію в більш простій і зрозумілій формі або навіть відчути інший час. На нашу думку, слово "відчувати" більш коректне, бо люди часто ототожнюють себе з героями книг або фільмів, намагаючись пов'язати свій досвід і досвід персонажа. У відеоіграх це відбувається набагато частіше і відчувається сильніше, бо гравець фактично управляє персонажем і простіше занурюється в історію, частиною якої він є. Завдяки відеоіграм люди вчяться і відкривають для себе нове, при цьому відпочивають, спілкуються, знаходять нових друзів.

Розробка ігор може виявитися не тільки захопливою, але і вельми прибутковою справою. Сам ринок відеоігор є масштабним і швидкозростаючим на цей час. Він привабливий для інвесторів великими прибутками, але при цьому вкрай складний для вибору проєкту для інвестиції через непередбачуваність. Успіх проєкту цілком залежить від того, чи сподобається контент користувачам чи ні, і як правило, вкладений бюджет ніяк не може вплинути на цей фактор.

Основна актуальність розробки відеоігор полягає в постійному попиті споживачів на нові продукти, тому що їм швидко набридає старе. Не зважаючи на те, що це відносно нове явище, вони вже давно стали в один ряд з фільмами та іншими розвагами для сотень мільйонів людей [2], для деяких і зовсім став основним заняттям у вільний час.

Пандемія коронавірусу викликала різке зростання продажів відеоігор і збільшила і так великий попит на їх розробку [3]. Самоізоляція дала людям поштовх привнести щось нове у своє життя.

Відеоігри стали настільки популярними, що на честь них почали проводити фестивалі, вести списки рекордів швидкісного проходження, кіберспортивні заходи які проводяться по всьому світу з багатомільйонними призами та величезною аудиторією. Появляються дослідження на тему відеоігор, нові способи як і на чому грати, попит на гарну графіку в іграх дає товчок для нових технологій, а гучні новинки часто призводять до обговорень в інтернеті. Кожен день виходить десятки нових ігор у різних жанрах: від звичайних пазлів до ігор з глибоким сюжетом, які ставлять перед собою ціль розказати про ту чи іншу проблему. На ринку є як маленькі, але амбіційні проєкти написані невеликими командами так і величезні франшизи від знаменитих розробників, це все завдяки прогресу, новим інструментам та креативним ідеям розробників, яких жадають гравці. Це, на перший погляд, просте і майже не значне заняття настільки ввійшло в наше суспільство, що вже складно представляти світ без нього.

Метою даної дипломної роботи є написання коду для гри. Гра була створена з використанням двигуна Unity, а весь інший контент для неї (моделі, іконки, музика, тощо) були створені з використанням безкоштовних програм або були взяті з вільних джерел і не підпадають під авторські права. Unity – це ігровий двигун випущений у 2005 році та постійно розвивається донині. Він дозволяє створювати різноманітні ігри, що працюють на персональних комп'ютерах, ігрових консолях і мобільних пристроях.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття Гри

Відеогра – це перенесення процесу гри в цифровий формат, але що з себе представляє саме поняття гри?

Гра, це в першу чергу, непродуктивна діяльність націлена на отримання того чи іншого досвіду, в якій важливий процес самої гри, а не її результат. Зазвичай гра є абстракцією реального світу або ж чогось відмінного, всі правила досить прості, а місце дії, як правило, обмежене. Як приклад, можна взяти всім відому гру в "морський бій", де гравці били навмання по клітинах противника, щоб знищити його флот. В цій грі в порівнянні зі справжніми бойовими діями максимально простий набір правил і прибрані майже всі нюанси.

Але крім тренувального аспекту можна виділити й інші властивості, ігри рятують від нудьги, розслаблюють і знімають стрес. Це пов'язано з їх властивістю затягувати увагу гравця, а відносна простота правил і передбачуваність дозволяють легко піти в стан потоку, при цьому приміряючи на себе якусь роль. Хоч і гра зазвичай не несе в собі чогось корисного, її аспекти можна впровадити в якусь продуктивну діяльність: наприклад за кожну виконану задачу нараховувати працівникам бали в загальній таблиці, тим самим підштовхнути їх на не явне суперництво і бажання набрати більше балів.

Як такої немає точної класифікації ігор через їх обширність та варіативність, але саме явище можна охарактеризувати таким чином:

– гра завжди має на увазі певну дію, наприклад фізичну, інтелектуальну, емоційну або соціальну;

– гра ініціюється внутрішньою потребою, наприклад, в навчанні або відпочинку;

– гра – це необов'язкове і свого роду непродуктивне заняття, тому що воно здійснюється в умовній і вигаданої ситуації.

1.1.2 Жанри комп’ютерних ігор

Ще 1958 році коли фізиком Вільямом Хігінботемом. була створена одна з перших відеоігор. Люди, тоді стояли поруч з симулятором тенісу на науковій виставці в Брукхейвенської національної лабораторії та навряд-чи б здогадувалися до чого це все приведе зараз. В ті часи обчислювальна потужність комп'ютерів сильно обмежувала можливості створюваної гри або програми, але тепер, коли технологічний прогрес зробив крок далеко вперед, розробники ігор можуть реалізувати все, що вони можуть замислити. Завдяки цьому за останні час з'явилося чимало нових жанрів і категорій, а розробники стали свого роду творцями, пробуючи нові підходи та втілюючи інноваційні ідеї. Прогрес дав новий і простий інструментарій для всіх, від величезних корпорацій до розробників одинаків втілюють свої мрії.

Головна відмінність між відеоіграми та звичайними іграми полягає в тому, що комп'ютерні ігри можуть самостійно обчислювати й підтримувати ігрові правила. Це дозволяє створювати більш складні та багаті ігрові світи. Також важливо сказати, що відеоігри ґрунтуються, в першу чергу, на ігровому процесі. На відміну від фільмів або літератури, в яких важливим аспектом є історія, в іграх важливий сам ігровий процес і інтерактивність. Хоча історія – це один з ключових компонентів, творці відеоігор в більшості зацікавлені в створенні хороших концепцій, які будуть захоплювати увагу гравця.

Завдяки доступності розробки комп'ютерних ігор до нашого часу їх стало багато, вони стали більш різноманітними та різнобічними. Ігри можна ділити за стилем, за необхідною швидкістю ухвалення рішень, за кількістю гравців в ній, за положенням камери, але основною є жанр гри. Критерії належності гри до того чи іншого жанру складно визначити однозначно через недостатню систематизованість самих комп'ютерних ігор в цьому плані. Через це навіть на різних сайтах або джерелах дані про жанр конкретної гри можуть відрізнятися. Проте, як раз ця нешаблонність і робить кожну гру єдиною у своєму роді. Розробники відеоігор вже давно прийшли до спільного консенсусу: майже завжди можна визначити, до якого з основних жанрів належить конкретна гра.

Жанр гри визначається самим ігровим процесом. Це те, як гравець взаємодіє з грою, і як гра реагує на ці взаємодії. Він є основним критерієм розподілу жанрів, оскільки це – основне, що відбувається в грі. Жанри ігор можна умовно поділити на три великі групи:

а) екшен ігри, основа яких є будь-яка дія, де упор ставиться на точність і швидкість її виконання;

б) ігри з явно вираженим стратегічним аспектом, де гра сильно залежить від виборів гравця;

в) Ігри, де взаємодія йде більшою мірою на рівні інформації, історії та взаємовідносин з цією історією.

Прекрасним прикладом жанром в групі екшенів буде Шутер, назва якого походить від англійського слова shoot, стріляти. В них більша увага приділяється стрільбі та точності гравця. Усередині жанру теж є свої важливі розмежування в залежності від положення камери, а саме шутери можуть бути від першої (FPS) або від третьої особи (TPS). Вид від першої особи показує те, що відбувається очима персонажа, за якого грають, в той час, як від третьої особи камера віддалена від героя, дозволяючи бачити більше, ніж в грі від першої особи. Наприклад, можна повернути камеру за кут або перешкоду, щоб побачити те, що відбувається за ними, а сам персонаж визирає з безпечного місця. Також обидва варіанти по різному впливають на занурення гравця в сюжет, в першому варіанті гравець простіше занурюється і передає персонажу свій характер, тим часом як у другому, спостерігаючи за персонажем і його поведінкою, можна дізнатися про нього більше.

Ще один приклад – жахи (Survival-horror). Ігри цього жанру притягують в основному атмосферою, вона в цих іграх є сумішшю постійної напруги, страху і тривоги. Вороги в таких іграх зазвичай з'являються несподівано, головний герой дуже слабкий фізично, а сама дія зазвичай відбувається в умовах поганої видимості, в тумані або вночі. Бувають випадки, коли гравець блукає в закритих просторах. Наприклад туман обігравався в грі Silent Hill в цій грі він створений не тільки з метою показати атмосферу міста, він також допоміг приховати головний графічний

недолік гри, а саме маленьку дальність промальовування об'єктів. На рис. 1.1.2.1 можна побачити те, як це виглядає в грі.

Рисунок 1.1.2.1 – Представник жанру жахів, гра Silent Hill.

Саме незнання того, що піднесе гра в найближчі 5 хвилин, тримає гравця в постійній напрузі. Якщо порівнювати ігри цього жанру з кінематографом, то можна знайти дуже багато спільного в стилі або подачі історії, але є одна суттєва відмінність: у фільмі глядач просто спостерігає, а значить, коли йому дуже страшно, може закрити очі та перечекати неприємний момент, в той час, як в іграх перечекати не вийде, бо сюжет і просування персонажа цілком залежить від гравця.

Стратегії унікальні тим, що часто в управління гравцеві дають не одного персонажа, а щось масштабніше, наприклад, управління містом або армію у військовій кампанії. У цих іграх гравцеві нерідко доступний цілий світ і всі його ресурси, внаслідок чого для перемоги необхідно застосовувати стратегічне мислення, щоб роздобути їх, правильно ними розпорядитися. Даний жанр як і

шутери можна поділити на 2 основні категорії: покрокові (TBS) і в реальному часі (RTS). Щоб з легкістю відрізнити ці два типи, потрібно просто подивитися, як йде ігровий час, наприклад якщо він ділиться на проміжки під час яких рішення приймають гравець або ШИ – це покрокова стратегія. Якщо всі рішення гравці роблять одночасно, в єдиному потоці часу, то це стратегія в реальному часі. Стратегії в реальному часі призначені для людей, які віддають перевагу динаміці дії та швидкість адаптації, в той час, як покрокові – для любителів продумати все до дрібниць, бо завжди є час оцінити ситуацію. Прабатьком жанру стратегій є традиційні настільні стратегічні ігри.

Основою рольових ігор (RPG) є відігравання ролі. Цей критерій дозволяє поставити її на ряд з інформаційними іграми. Гравцеві дається під контроль певний персонаж, готовий або створений гравцем, з початковим набором навичок, характеристик, вмінь, які в міру гри будуть удосконалюватися. Рольові ігри, це в першу чергу, історія про розвиток персонажа і його шляху становлення, про спілкування з навколишнім світом, тому при розробці гри в цьому жанрі грамотне опрацювання сюжету та ігрового простору є дуже важливою частиною. Хороша рольова гра на думку гравців повинна мати глибокі діалоги, пророблений сюжет, багато різних кінцівок, великий і цікавий ігровий світ відкритий для дослідження.

Як наприклад в грі Vampire: The Masquerade - Bloodlines, де гравцеві потрібно взяти на себе роль вампіра. Вона вважається культовою серед ігор цього жанру на думку багатьох гравців і була побудована на однойменній настільної рольової грі. Спочатку гравцю дають можливість вибрати клан і стать персонажа, а також розподілити певну кількість очок між та навички та атрибутами (вони зображені на рис. 1.1.2.2). Всі ці параметри можуть кардинально змінити стиль проходження гри. Так, створивши персонажа Малкавіана гравець буде постійно стикаються з галюцинаціями, незвичайними діалогами і ситуаціями, недоступними для інших кланів.

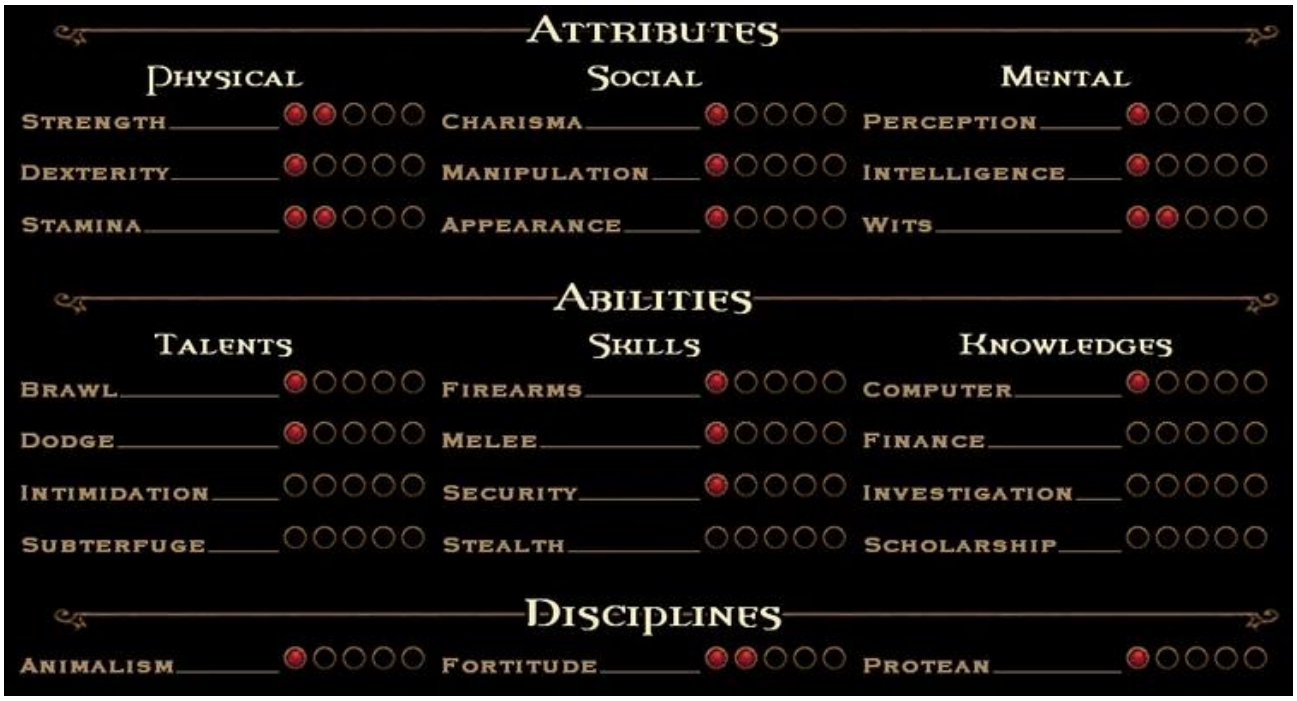


Рисунок 1.1.2.2 – Атрибути та характеристики персонажа в RPG грі Vampire: The Masquerade - Bloodlines.

У масових багатокористувацьких рольових ігор (MMORPG) гравці взаємодіють не тільки з контентом гри, а й один з одним. Соціальна взаємодія між гравцями тут є основою гри: їм доводиться збиратися в клани та альянси, битися між собою за землі та замки, разом ходити в складні підземелля і полювати на сильних монстрів, торгувати на загальному ринку. Зазвичай в таких іграх в одному світі можуть грати сотні та тисячі людей.

1.1.3 Етапи розробки.

Створення відеоігор є дуже тривалим і трудомістким процесом, який містить в собі як технічні, так і творчі аспекти. Найчастіше, над однією відеогрою працює велика команда, де кожна людина є фахівцем в тій чи іншій області. Гра – це спільна праця митців, програмістів, звукорежисерів, дизайнерів, письменників, маркетологів та інших людей. Час розробки сучасної гри в середньому займає від одного до трьох років, хоча бувають винятки пов'язані зі складністю розробки. Є проєкти, які не просто довго роблять, їх ніяк не можуть завершити, зрушити з мертвої точки. Таке буває, наприклад коли ідея проєкту настільки інноваційна, що

за нею не встигають технології. Ще одна проблема, яка може перешкодити розробці, - це не до кінця сформована ідея або усвідомлення, що вона не те, що потрібно цьому проєкту. Крім усього цього є і банальні проблеми з графіком праці, непередбачені ситуації, різна думка розробників на ті чи інші аспекти.

Способи підходу до розробки можуть змінюватись від команди до команди. Наприклад в одній команді всі розробники можуть пропонувати свої ідеї для реалізації, а в іншій цим займається виключно гейм-дизайнер, але основна хронологія етапів розробки в цілому скрізь однакова.

Етапи розробки можна розділити на 4 частини: концепція, підготовка до виробництва, саме виробництво і поствиробництво [4]. На кожному з цих етапів працюють певні групи фахівців, на відміну від дизайнерів, які обов'язково занурені в усі етапи. Це ті людей, які придумують гри та стежать за їх розробкою, контролюючи всі аспекти від механік до зовнішнього вигляду гри.[5]

Концепція – це етап створення та опрацювання ідеї гри. Насамперед розробники як генератори ідей вирішують до якого жанру буде ставитися гра, що в ній буде присутнє, які ідеї використовувати, проводять початкове опрацювання ігрового дизайну. Мета даного етапу полягає в створенні гейм-дизайнерської документації. Він служить для закріплення і передачі ідей і дизайну гри розробникам і іншим зацікавленим особам, наприклад видавцям. Ця документація складається з різних документів, кожен для своєї мети. Перш ніж приступити до його створення, потрібно розуміти, для чого і кого він пишеться. Це допоможе правильно формулювати думки та підбирати потрібні вирази, щоб ідея укладена в документі правильно доходила до читача. Якісна та продуктивна документація повинна бути структурованою, мати повний опис продукту і підлягати регулярній актуалізації.

Кожна ідея, яка подумки здається хорошою, вимагає перевірку в реальності, тобто створення прототипу. Прототип створюється для оцінки ігрового процесу, перевірки різних ідей і тестування ігрових механік. На етапі створення прототипу потрібно реалізовувати тільки те, що потрібно перевірити в стислі терміни. По суті це свого роду чернетка для розробників.

Після того, як ідея буде сформована і перевірена, потрібно визначити що і як буде використовуватися для її реалізації. На цьому етапі також вирішують, на якому двигуну гра буде працювати. Художники створюють концепцію і стиль для гри, який був задуманий, а сценаристи пишуть історію. На даному етапі створюються окремі шматочки гри, які будуть зібрані в майбутньому.

Після створення і перевірки ідеї починається найскладніший етап - виробництво. Протягом цього етапу йде вся основна розробка гри. У цей момент всі члени команди починають займаються тією роботою, яка була їм визначена. Створюються дизайн, все наповнення світу, йде програмування, записується звук. Вся ця робота відбувається під наглядом основних розробників та дизайнерів за складеним планом.

Після того, як гри була розроблена починається етап поствиробництва. На цьому етапі починається перевірка гри й подальше її вдосконалення. Тестувальники повинні грати в неї та знаходити всі можливі помилки й недоробки, писати про них звіти та передавати розробникам для подальшого виправлення. Не всі помилки можуть бути знайдені відразу: про деякі з них дізнаються тільки після того, як гра потрапить в руки до гравців. У таких випадках розробники випускають нові обновлення з виправленням даних помилок, які гравці зможуть завантажити через інтернет. Проводиться фінальні налаштування графіки, ігрового процесу та інших дрібниць і починається підготовка до запуску, дороблюється все, що можна доробити. Паралельно на цьому етапі гру демонструють людям, викладають в мережу трейлери або навіть демоверсії, одним словом привертають увагу на проєкт. Також ймовірно що навколо конкретної гри або серії утворюються ігрові спільноти або інтернет-ресурси. Подібні групи зацікавлених людей генерують великий потік інформації, який допомагає у просуванні гри.

1.2 Опис проєкту

1.2.1 Загальні відомості

Суть гри: два гравці будуть пересуватися по нескінченному і випадково генерованому треку. На ньому можна їздити як зверху так і знизу "догори ногами". Щоб перевернутися на іншу сторону треку потрібно виїхати за його межі в порожнечу - машина автоматично перевернеться і змінить свою гравітацію. Трек ділиться на дві лінії які генерується незалежно одна від одної. Лінії діляться на сегменти, вони можуть бути прямими або похилими. Похилі сегменти зроблені для зміни положення лінії у висоті, вони одночасно є підйомом з одного боку і спуском з іншого. За правилами гравець не може підійматися, щоб пройти таку перешкоду йому потрібно перевернутися на інший бік треку, де даний перепад буде вже спуском, або ж перескочити на іншу лінію якщо у нього є така можливість. Для кращого розуміння ідеї варто поглянути на візуалізацію, на рис. 1.2.1.1 зображений прототип треку і дві машини гравців на ньому.

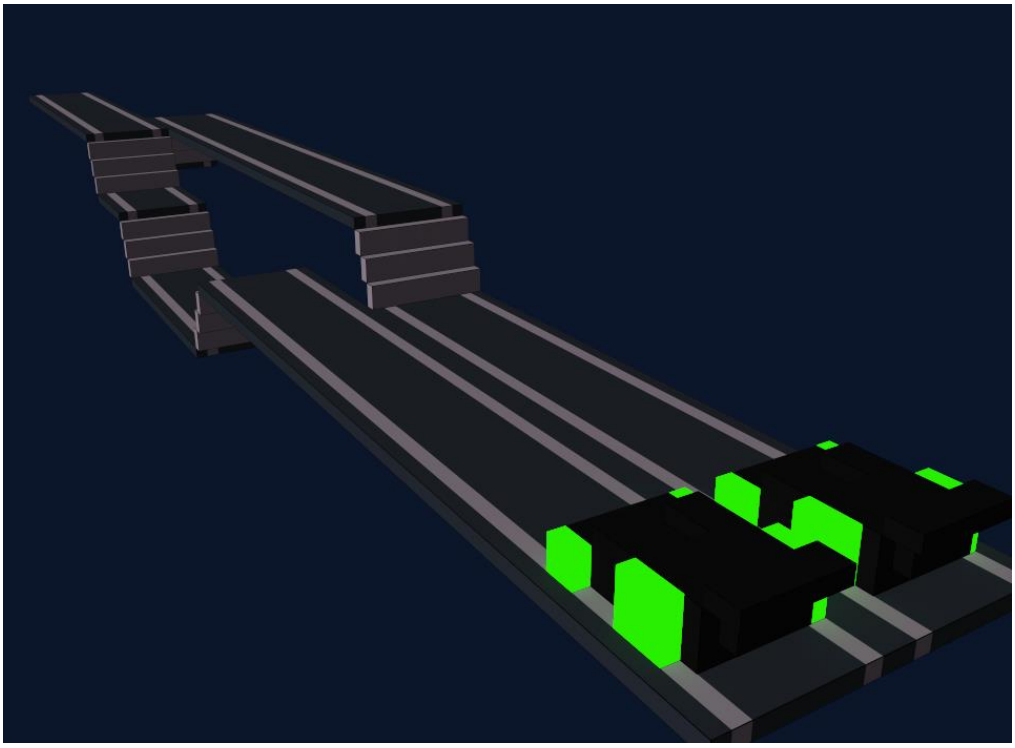


Рисунок 1.2.1.1 – Прототип треку

Також не упускається можливість того що гравець втратить управління і не встигне повернутися на трек при спробі перевернути машину. В такому випадку гравець випадає за межі треку і також програє. Перегони тривають поки один з гравців не вріжеться або не вилетить з треку, той хто що залишився в живих стає переможцем. Гравці не можуть впливати на машини один одного.

1.2.2 Діаграма варіантів використання

В ході аналізу вимог до проєкту була розроблена UML-діаграма варіантів використання, Вона відображена на рис. 1.2.2.1 . Одна копія гри розрахована на одного гравця, після запуску програми гравець знаходиться в головному меню з якого можна налаштувати гучність звуку і мову або почати пошук гри. У разі успішного пошуку опонента гравець разом з противником переходять на трек і отримує в керування машину.

Машина керується стрілками на клавіатурі й постійно рухається вперед, з часом збільшуючи швидкість. Якщо машина гравця врізається в перешкоду або випадає за межі треку то гра закінчується, а гравець вважається таким, що програв. Після закінчення гри гравець може повернуться до головного меню.

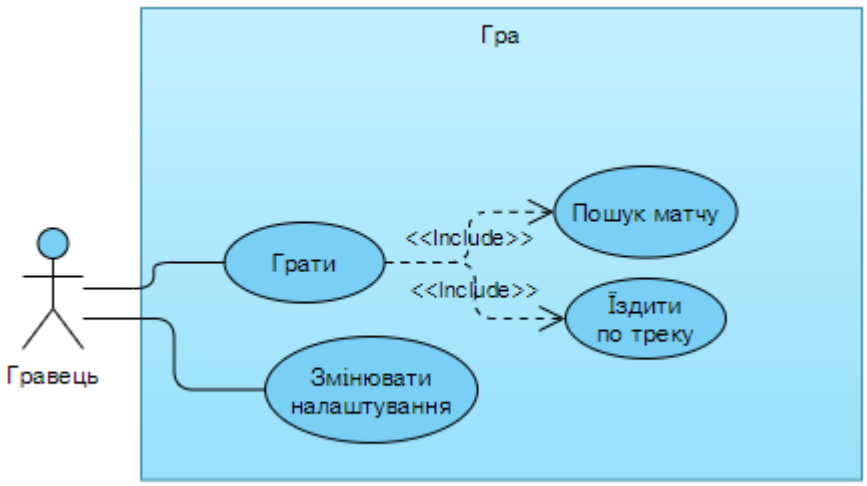


Рисунок 1.2.2.1 – Діаграма варіантів використання

1.3 Аналоги

Основною частиною ідеї проєкту є нескінченна і випадково генерована карта, у більшості проєктів дана ідея зустрічається дуже рідко. Саме цей елемент відрізняє продукт від інших на ринку у цьому жанрі. Аналоги та їх порівняння представлені в таблиці 1.3.1.

Таблиця 1.3.1 – Порівняння проєкту з аналогами.

Назва	Характеристики		
	3D графіка	Багатокористувацький режим	Процедурна генерація дороги у грі
FliperDrive (Цей проєкт)	так	так	так
Asphalt 9	так	так	ні
Earn to Die 2	ні	ні	ні
Distance	так	так	ні
Traffic Racer	так	ні	так

Аналоги наведені в цій таблиці вибиралися серед продуктів для різних платформ, тут присутні як прості в управлінні та візуальному плані, так і добре опрацьовані проєкти, але в цілому у всіх простежуються риси жанру аркадних гонок.

2 АНАЛІЗ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ

2.1 Дослідження існуючих інструментальних засобів

Для розробки ігор можна видати такі засоби:
а) Ігровий двигун – це програмний комплекс, який розширює розробку ігор, надаючи розробник набору необхідних інструментів. Як правило, в себе входять компоненти, що відповідають за візуалізацію, фізичні обчислення, звук, системні скрипти, анімацію, мистецький інтелект, мережевий код, управління пам'яттю та багатопоточність. Двигун це основна частина, ядро гри, його завдання полягає в організації та обробці ігрових об'єктів, а також для їх зображення на екрані. Існує величезна кількість ігрових двигунів, навчальних від усіх відомих брендів, кінцевих тем, які розробники створюють спеціально для своїх проєктів.
б) Конструктор ігор – об'єднує в собі ігровий двигун і середовище розробки, спрощує процес створення ігор. Його завдання у швидкому створенні комп'ютерних ігор, людям без спеціальної підготовки. Буває так що вони обмежені певними жанрами або дають можливість створювати тільки 2D гри
в) Також існують Фреймворки - це який-небудь набір бібліотек або модулів, який облегшує розробку, що дає певний початковий функціонал і структуру для проєкту. Користувачу пропонується його розширити до досягнення необхідного результату. Нижче будуть розглянуті одні з найпопулярніших представників даних груп.

2.1.1 Unity

Unity це величезне середовище для розробки комп'ютерних ігор.
У ньому є різні програмні засоби необхідні при створенні гри. Компілятор, налагоджувач, ресурси для роботи зі спрайтами та багато іншого. Завдяки зручності використання, і компонентно-орієнтованого підходу Unity дозволяє створювати гри максимально простим, комфортним і швидким чином.

В рамках цього походу розробник створює об'єкти до яких додає різні компоненти, свого роду властивості. Завдяки функціональним графічного

редактора з системою Drag & Drop двигун дозволяє з легкістю створювати карти та розставляти об'єкти. Вигляд редактора показаний на рис. 2.1.1.1.

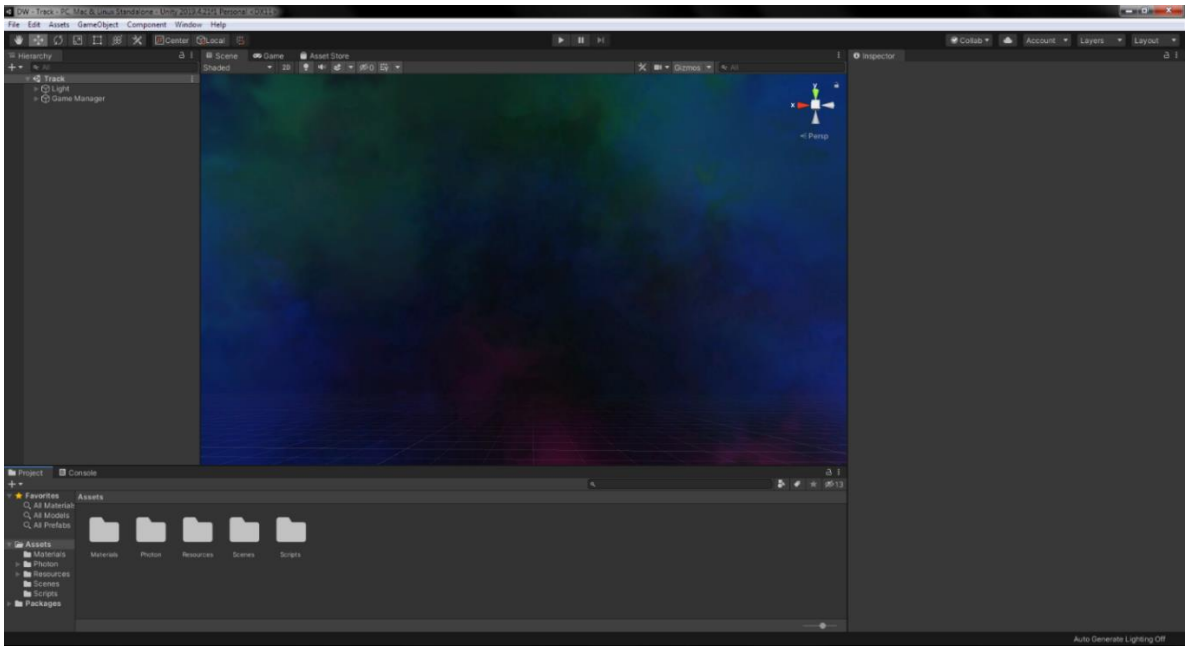


Рисунок 2.1.1.1 – редактор Unity.

Безумовно великим плюсом є наявність величезної бібліотеки ресурсів під назвою Unity Asset Store, за допомогою яких можна значно прискорити процес розробки гри. Велика кількість користувачів завантажують свої напрацювання і ресурси які можна використовувати у своєму проєкт. Багато ресурсів доступні безкоштовно, але є й ті, які пропонуються за невелику суму, при бажанні можна створювати власний контент і публікувати його на майданчику Unity Asset Store.

За допомогою цього ігрового двигуна можна розробляти не тільки додатки для комп'ютерів на базі Windows, Mac або Linux, але і для мобільних пристроїв Android або IOS, ігрових приставок і інших девайсів включаючи веб плагіни. На противагу плюсів з кросплатформенності та універсальності можна поставити проблеми зі швидкодією. На Unity можливо але складно зробити великий і в той час стабільний проєкт, він прекрасно підходить для невеликих розробок.

У зв'язку з розмірами проєкту мінуси цього двигуна не настільки критичні та навряд чи зашкодять йому.

Перша версія Unity з'явилася у 2005 році, коли ігровий двигун був анонсований на Worldwide Developers Conference. Unity спочатку призначався для комп'ютерів на базі Mac [6]. Однак розробники чудово розуміли, що частка цієї ОС на ринку дуже мала, тому перше глобальне оновлення яке з'явилося в серпні додало можливість збірки ігор під Windows. Незабаром додали ефекти постобробки та тіні, вбудований скрипт управління персонажем, розширили можливості редактора скриптів, виправили безліч помилок.

Версія 2.0 мала більш ніж 50 новими функціями такими як оптимізований двигун ландшафту або динамічні тіні які опрацьовувалися в реальному часі з підтримкою точкових джерел світла [7]. Основною і важливою зміною стало додавання повноцінної підтримки редактору для Windows і підтримку бібліотек DirectX.

Третя версія вийшла у 2010. Зі змін додали можливість міняти місцями всі елементи редактора, поліпшили карти освітлення, додана можливість відкладеного рендеринга (збільшення ефективності обробки джерел світла, за допомогою відділення розрахунку геометрії сцени від прорахунку освітлення), і візуалізацію тільки видимих на екрані об'єктів, і інше безліч нововведень націлених на оптимізацію і якісну роботу зі звуком [8].

Unity 4.0 вийшла в листопаді 2012 дала можливість працювати з ним під операційною системою Linux а також отримала підтримку DirectX 11. Пізніше оновлення 4.3 представило нові інструменти для розробки для 2D-ігор [9]. До впровадження інструментів для 2D розробникам доводилося йти на різного роду хитрощі при створенні своїх проєктів, наприклад використовуючи майже плоскі прямокутники як аналоги теперішніх спрайтів.

Unity 5.0 привнесла підтримку Vulkan, він збільшує швидкість обробки при одночасному зниженні навантаження на драйвери та процесор. Крім нього було додано безліч функцій, це, мабуть, один з найбільших релізів за весь час існування Unity. Це оновлення торкнулося майже все: від створення повноцінний звукового редактора до доопрацювання 2D фізики та фізично коректних матеріалів. На рис.

2.1.1.2 показані порівняння з попередніми версіями в області якості відображення м'якої тіні від направленого світу [10].

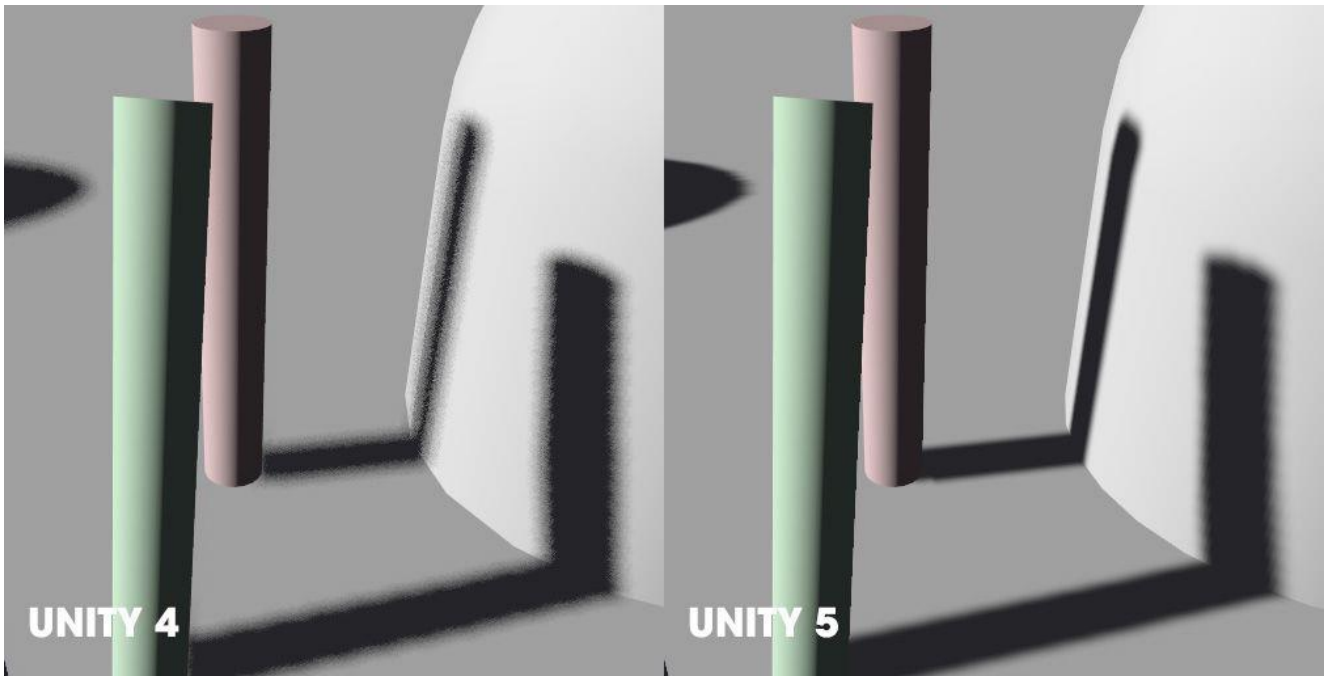


Рисунок 2.1.1.2 – Порівняння тіней в версіях Unity 4 та 5.

Надалі Unity змінила систему нумерації версій, тепер вона будувалася не просто на порядковому номері, а на основі дати. 2017 версія [11] виконав колосальну роботу в плані фізики та роботи зі спрайтами, наприклад для останніх поліпшили роботу атласів і додали маски. Unity розвивається дуже швидко і з кожним релізом намагається охопити все більш і більш інноваційні функції. Версія 2019 отримала чергове оновлення інтерфейсу. і новим інструментом Visual Effect Graph. Це редактор візуальних ефектів на основі вузлів, він дозволяє створювати візуальні ефекти які Unity імітує безпосередньо на графічному процесорі. Крім того, Unity з'явилася система трасування променів. [12]

В Unity 2020 був інтегрований до цього платний інструмент Bolt, призначений для візуального програмування. Основна мета візуального програмування полягає в спрощенні розробки, по суті з ним користувачі можуть розробляти логіку для гри без необхідності писати код. Unity придбала Bolt, а після він був включений в усі тарифні плани ігрового двигуна [13].

У Версії за 2021 рік була виділена велика увага підвищенню стабільності, і ряду значущих поліпшень для всього движка. Також поліпшення торкнулися раніше згаданий Bolt.

2.1.2. Unreal Engine

Unreal Engine випущений в 1996 році та розроблений та підтримуваний компанією Epic Games. У ньому використовується C ++ та Blueprint — редактор візуального скрипту (Вигляд якого зображений на рис. 2.1.2.1), який часто використовується для швидкого прототипування та роботи з рівнями.

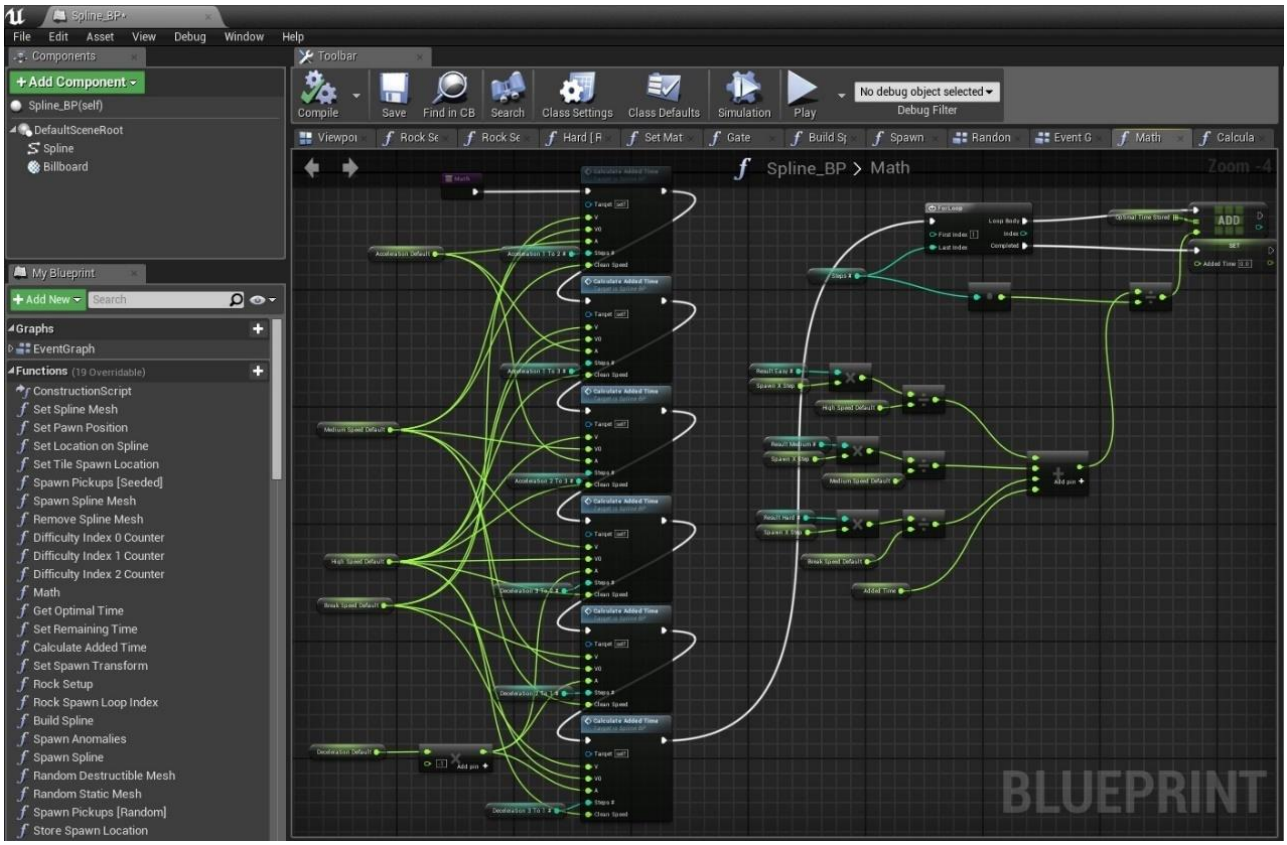


Рисунок 2.1.2.1 – приклад вікна для роботи з візуальним програмуванням в Unreal Engine за допомогою Blueprint.

Основна перевага цього ігрового двигуна полягає в якісній графіці, моделюванні моделей та дизайні рівнів. Двигун популярний у 3D-художників,

моделей та левел-дизайнерів. Також він славиться м'якими тінями, якісним освітленням достовірною анімацією персонажів та іншими ефектами.

Внаслідок цього, двигун також використовується у створенні комп'ютерної графіки в кіноіндустрії. Є підтримка різних системних рендерінгів серед яких є Direct3D, OpenGL. Для ігор через мережу підтримуються технології Windows Live, Xbox Live та GameSpy, які дозволяють одночасно підтримувати підключені 64 ігрові системи.

Він більшою мірою призначений для більших проєктів класу AAA, що розроблені професіоналами, з цього виказує, що він сам по собі важкий і не ставить перед собою просто у використанні. В UE невелика документація та навчальна база. З 2015 року безкоштовний для використання. Якщо гра приносить понад 3000$ за квартал, виробники беруть 5% прибутку.

2.1.3 Godot

Godot Engine, випущений у 2007 році, вперше зарекомендував себе як прекрасний 2D-двигун, а також має підтримку 3D. Основні мови програмування - це GDScript і C #, а також візуальний скрипт для чого-небудь простого.

У Godot відомий своєю древовидною структурою елементів. На практиці це означає, що нові об'єкти на рівні додаються як вітки, які вже існують на нім вузлам і в будь-який момент самі можуть стати повноцінною сценою. Графічна система для всіх платформ побудована на OpenGL ES 3.0. Для створення шейдерів використовується узагальнена шейдерна мова, яка є близькою до мови GLSL. Є можливість повноцінного створення шейдерів у візуальному редакторі.

Двигун не вимогливий до ресурсів комп'ютеру і створює гарну картинку. Готових трьохмірних інструментів не багата кількість, але достатньо для реалізації багатьох ідей. На рис. 2.1.3.1 можна побачити вид редактора при роботі у режимі 3D, у нього можна перейти за допомогою кнопки в самій верхній панелі посередині.

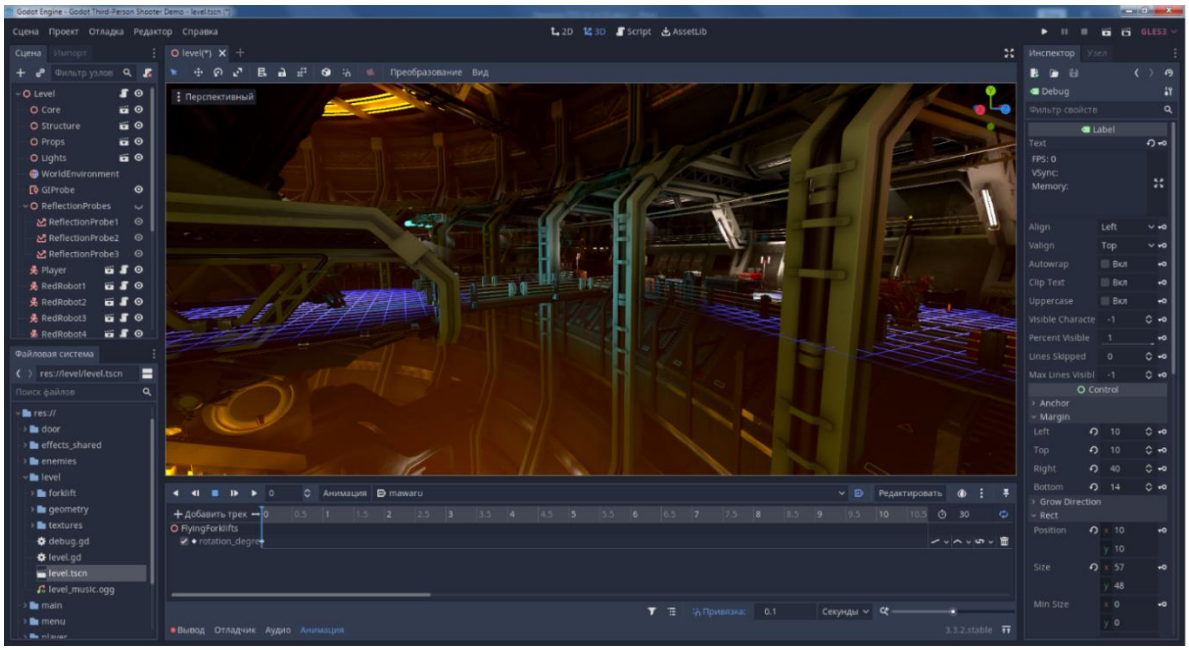


Рисунок 2.1.3.1 – вікно редактора Godot Engine.

Godot максимально полегшений і не вимагає установлення на комп'ютер, він дуже простий у використанні та має хорошу навчальну базу [14]. Він давно завоював звання самого дружнього та легкого у освоєнні інструментів і є повністю безкоштовним.

2.1.4 CryEngine

CryEngine був створений німецьким підрозділом компанії Crytek. Він використовується у всіх іграх, розроблених самою Crytek. Даний ігровий двигун не універсальний, він створений для жанру шутерів.

Він також відомий своїми естетично приємними та якісними іграми, моє підтримку DirectX 12, Vulkan API, VR, [15]. З часом вони оновлюють свою систему, щоб підтримувати нові консолі. Для розробки використовуються C ++, C # і lua, візуальний редактор. CE дуже потужний ігровий двигун у графічному плані, який дозволяє створювати ігри з майже фотореалістичною графікою.

CryEngine буде хорошим вибором для певних груп людей із визначеними цілями. Починаючи з 2016 року двигун і набір засобів розробки,

розповсюджуються безкоштовно, але за умови виплати Crytek 5% прибутку при доходах, котрій перевищують визначений ліміт.

2.1.5 Source

Source 2 був розроблений компанією Valve. Розробка першої версії двигуна почалася ще в 1998 році, коли Valve підходила до фінального етапу створення Half-Life. У першу чергу Source хорошій своєю доступністю. Інструменти Source SDK доступні всім користувачам Steam. Цей набір містити редактор карт, програми для імпорту, експорту та перегляду ресурсів, а також програми для розпакування файлів та файлів вихідного коду бібліотеки для деяких ігор Valve. Цей набір можна використовувати як для створення модифікацій, так і окремих ігор. Розробка на Source вимагає хорошого знання C ++. На рис. 2.1.5.1 показаний редактор цього двигуна.

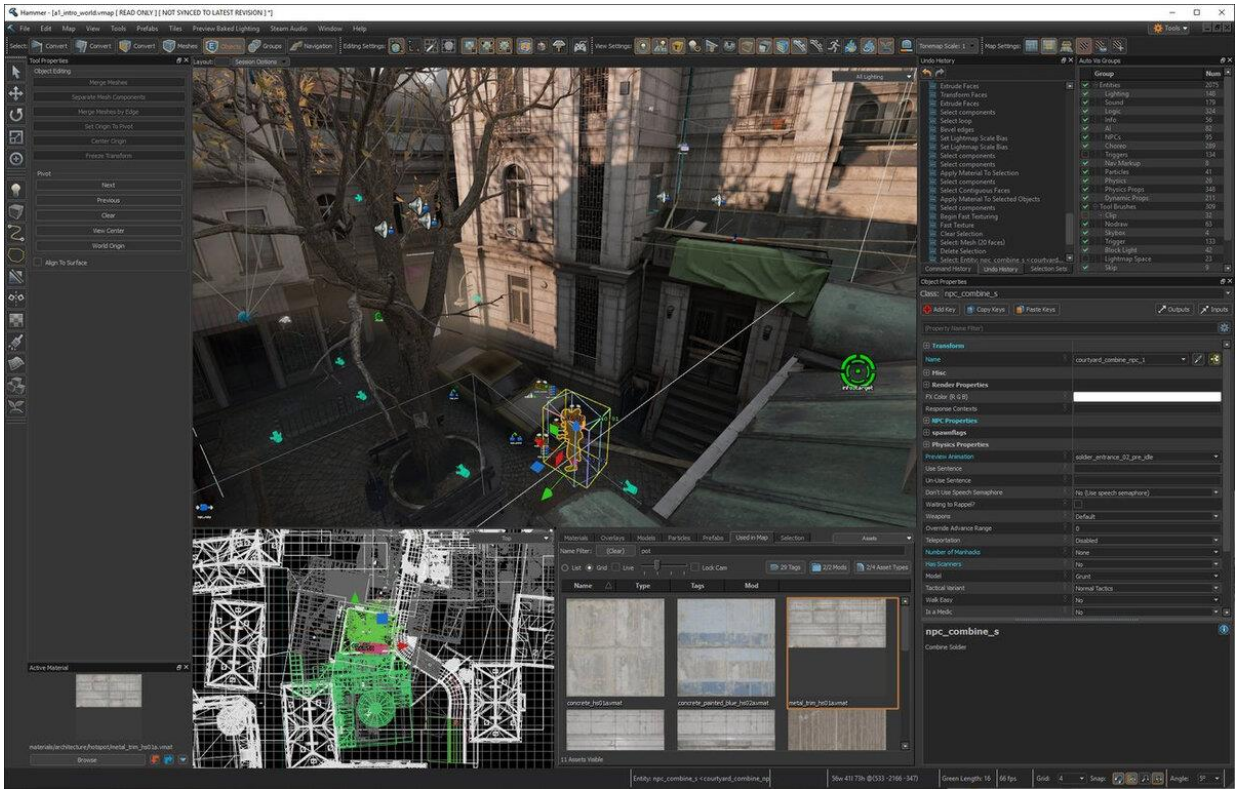


Рисунок 2.1.5.1 – вигляд редактору Source.

Даний двигун має перевагу в інтегрованій інтеграції з Майстерню Steam, це дозволяє, наприклад, з легкістю створювати та додавати моди в іграх на денному двигуну. На жаль двигун не може конкурувати з іншими конкурентами на ринку через малої кількості можливостей. Якщо розробник вирішить продати свій продукт, за ліцензійним погодженням він повинен завантажить його також і в Steam [16]

2.1.6 Unity photon

Photon Unity Networking (PUN) не є редактором. Це хороший інструмент у створенні логіки для багатокористувацької гри на базі Unity.

Якщо порівнювати PUN і стандартний Unity Networking, то хоч і обидва вони мають архітектуру сервер / клієнт, PUN своєю чергою підтримує однорангове посилання пакетів. Тобто в той час, як в Unity Networking всі повідомлення повинні проходити через хост-клієнт і не можуть бути відправлені безпосередньо між вузлами, в PUN пакети направляються в ретранслятор, а від нього до адресатів, також є можливість повністю обійти ретранслятор, і клієнти можуть зв'язуватися безпосередньо, тим самим зменшуючи кількість переходів з двох до одного. Також з мінусів Unity Networking можна виділити те що у випадку виходу хост-клієнта з мережі гра зупиняється. PUN дає можливість передавати звання хост-клієнта на льоту.

Всі обробки по підключенню і синхронізації відбуваються на хмарному сервері Photon Cloud. Для даної гри буде використовуватися безкоштовний тарифний план розрахований на 20 осіб одночасного підключення, в разі потреби можливо покупка додаткових потужностей до 50000 підключень[17]. На етапі розробки стартового тарифного плану цілком вистачить.

2.2 Порівняння інструментальних засобів

Для полегшення аналізу потрібно зібрати дані по доступності, актуальності, функціональних можливостей. Данні будуть зібрані з офіційних порталів та ресурсів інструментальних засобів вигляді таблиць (2.2.1 та 2.2.2).

Таблиця 2.2.1 – Порівняння доступтоності та актуальності інструментальніх засобів.

Назва	Доступність	Дата останнього релізу
Unity	умовно безкоштовно	23.3.2021
Unreal Engine	умовно безкоштовно	3.12.2020
CryEngine	безкоштовно	30.7.2020
Sourse 2013	безкоштовно	10.9.2015
XNA	безкоштовно	12.3.2021
Construct 3	безкоштовна тріальна версія	11.5.2021
Game Maker	безкоштовна тріальна версія	16.4.2021
Amazon Lumberyard	безкоштовно	20.5.2021

Таблиця 2.2.2 Аналіз функціональних можливостей.

Назва	Графіка		Мова програмування
	2D	3D	
Unity	так	так	C#, JavaScript

Продовження таблиці 2.2.2

Назва	Графіка		Мова програмування
	2D	3D	
Unreal Engine	так	так	C++, BluePrint
CryEngine	ні	так	C++, C#, lua
Sourse 2013	ні	так	C++
XNA	так	так	C#
Construct 3	так	так	JavaScript
Game Maker	так	так	GML
Amazon Lumberyard	ні	так	C++, lua

Для порівняння популярності буде братися динаміка з інструменту Google trends. Цей інструмент дозволяє безкоштовно аналізувати сезонність і тренди та порівнювати популярність пошукових запитів на різних мовах практично в реальному часі. Було взято 5 найпопулярніших двигунів зі списку віще, а саме Unreal Engine, CryEngine, Game Maker, Construct та Unity і порівняно у цьому інструменті. На графіку шкала інтересу зображена в співвідношенні від 0 до 100. На рис. 2.2.1 можна помітити що ігровий двигун Unity має явну перевагу в кількості пошукових запросів по всьому світу за минулий рік.

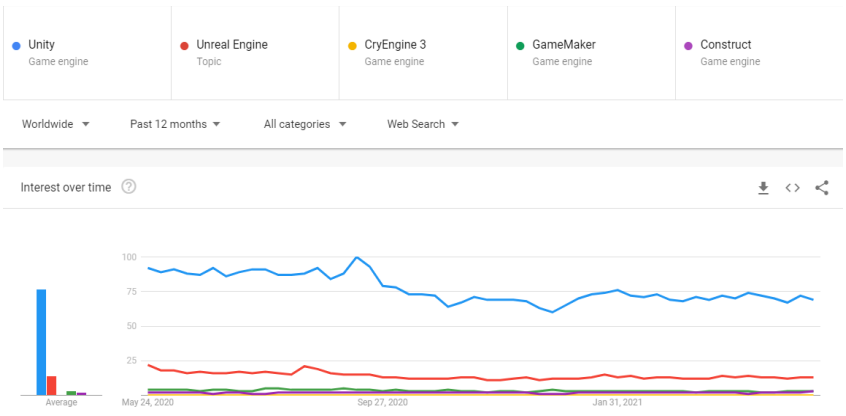


Рисунок 2.2.1 – Динаміка популярності.

Популярність є важливим фактором: чим більше спільнота користувачів двигуна, тим більше навчального матеріалу, інструментів чи доповнень для двигуна воно створює. Ще одним плюсом спільноти є підтримка, адже можливість порадитися з іншими людьми в питаннях де розробник не компетентний теж важливо. Дивлячись на тенденції ігрового ринку можна сказати що двигун не визначає успіх чи невдачу гри, це інструмент який впливає тільки на етап розробки, тому розробники намагаються вибирати для себе найбільш універсальний і швидкий інструмент. Настільки велику популярність Unity можна пояснити його універсальністю, великою кількістю інструментів і низьким порогом входження. Дані якості виділяють його серед конкурентів на ринку для початківців одинаків і досвідчених студій.

2.3 Висновок розділу

Ігрова індустрія як і інструменти для розробки ігор розвивається з неймовірною швидкістю. За недовгий час ця галузь розвинулася від 8-бітових ігор запускаються на ігрових приставках підключених до телевізора до великих проєктів які використовують технології віртуальної реальності. На цей час існує величезна кількість ігрових двигунів, різних типів і напрямків, одні із найпопулярніших були описані в цьому розділі.

З огляду на наведені таблиці, графіки, і опису популярних ігрових двигунів був зроблений вибір у сторону Unity. Даний двигун підходить для поставлених в дипломній роботі цілей, умовно безкоштовний і підтримує потрібні компоненти включаючи мову програмування C #. Також завдяки своїй гнучкості й підтримки багатьох популярних платформ дозволяє в майбутньому переносити проєкт на інші операційні системи. Також через популярність самого двигуна програмісти які працюють з ним завжди у попиту на ринку.

3 ПРОЄКТУВАННЯ ТА СТВОРЕННЯ ПРОГРАМНОГО ПРОДУКТУ

3.1 Архітектура

Проєкт в Unity складається зі сцен (Scene), на яких розташовані ігрові об'єкти (GameObject) з прикріпленими до них компонентами (Component). У кожного об'єкта є обов'язковий компонент Transform, який відповідає за розташування об'єкта на сцені. Також, можуть бути підключені додаткові компоненти, як готові, так і створені користувачем. [18]

Програмування на даному ігровому движку полягає в створенні призначених для користувача класів які підключаються до об'єктів у якості компонентів. Всі призначені для користувача класи повинні успадковуватися від класу MonoBehaviour. Для обробки багатокористувацької логіки потрібно наслідувати клас MonoBehaviourPunCallbacks, це той самій клас MonoBehaviour, але з вбудованими методами зворотного виклику бібліотеки PUN. Також є IPunObservable який відповідає за спостереженням і синхронізацією обраних параметрів в класі. У самому пане вже є компоненти з допомогою яких можна синхронізувати розташування, поворот і розмір об'єкта, але для синхронізації інших параметрів потрібно створити окрему логіку для посилання і приймання пакетів. Файли з класами прийнято називати скриптами.

3.2 Діаграма класів

Для реалізації даного проєкту була побудована діаграма класів, в ній будуть присутні:

1) Клас GameLogicManager відповідатиме за загальну логіку під час гри, а саме перевірку стану машин і перемикання гри між етапами в залежності від їх стану.
2) Клас TrackGenerator відповідає за генерацію треку, його методи будуть обчислювати положення і створювати ділянки дороги.

3) Клас LobbyManager відповідатиме за обробку кнопок головного меню, початкові значення параметрів мові та гучності звуку і підключення гравців до сервера і між собою.

4) Клас PlayerControls відповідатиме за управління машини гравця і його поведінку в тих чи інших ситуаціях, таких як переворот або зіткнення з перешкодою. Також тут буде оброблятися UI під час гри.

5) Клас SelfDestruction буде знищувати непотрібні об'єкти на карті, ті які гравці вже проїхали.

Безпосередньо саму діаграму можна побачити на рис. 3.2.1

.

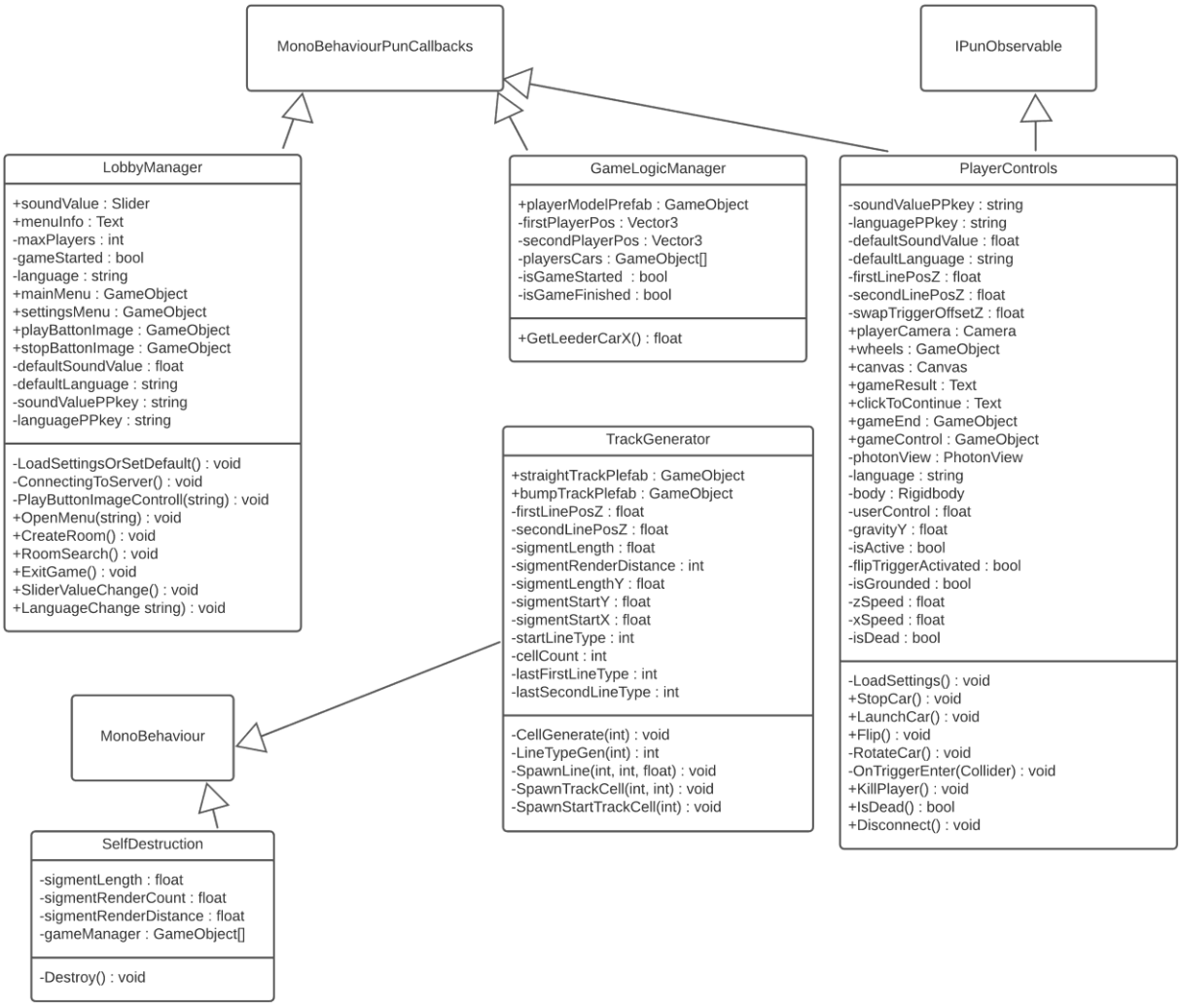


Рисунок 3.2.1 – Діаграма класів.

3.3 Основна ігрова логіка

При появі на треку для гравця створюється його машина управляючи якою він буде їздити на треку. Ігрову сесію можна умовно поділити на 3 етапи: підготовка, поїздка, фінал. При підготовці йде очікування завантаження всіх гравців, машини не зрушать з місця поки все гравці не завантажаться. Після вдалого завантаження гравців усі машини одночасно стартують. Протягом поїздки йде перевірка на наявність врізалися машин і збереження координат поточного положення гравців які буде використовувати скрипт генерації карти. Після того як один гравець програє гра зупиняється, спливає вікно з результатами та пропозицією повернуться в головне меню.

За оброблення цієї ігрової логіки відповідає об’єкт GameManager який розташований на ігровій сцені, до нього прикріплені скрипти GameLogicManager і TrackGenerator, що показано на рис. 3.3.1.

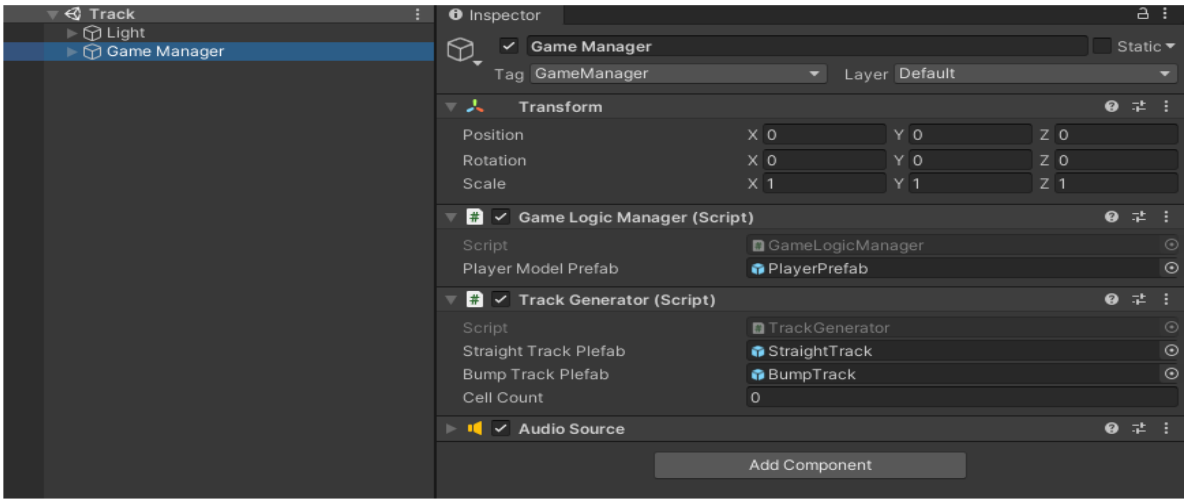


Рисунок 3.3.1 – Об'єкт GameManager у вікні редактора Unity.

Основний ігровий цикл очікування, старту і перевірки стану гравців відбувається в GameLogicManager в методі Update який був успадкованим від MonoBehaviour. Update викликається один раз за кадр після запуску скрипта і протягом всієї його активності. за допомогою нього можна проводити

перевірки або обчислення на постійній основі. Наприклад створивши об'єкт і додавши до нього скрипт в коді якого в Update до поточного повороту об'єкта буде додаватися пів градуса, то запустивши проєкт можна побачити як даний об'єкт обертається. На рис. 3.3.2 показаний код всередині GameLogicManager в методі Update.

```
void Update()
{
    RenderSettings.skybox.SetFloat("_Rotation", Time.time * 0.4f);

    if (!isGameStarted)
    {
        playersCars = GameObject.FindGameObjectsWithTag("Player");
        if ( PhotonNetwork.PlayerList.Length == playersCars.Length)
        {
            // starting when all players load ////////////////////////////////////////////
            isGameStarted = true;
            foreach (var obj in playersCars)
            {
                obj.GetComponent<PlayerControls>().LaunchCar();
            }
        }

    }
    else
    {
        if (!isGameFinished)
        {
            transform.position = new Vector3(GetLeederCarX(), 0, 0);
            foreach (var obj in playersCars)
            {
            if (obj.GetComponent<PlayerControls>().IsDead()) { StopGame(); }
                else
                {
                    // if all players alive ////////////////////////////////////////////
                    if (obj.transform.position.x > transform.position.x) transform.position = new Vector3(GetLeederCarX(), 0, 0);
                }
            }
        }
    }

}
```

Рисунок 3.3.2 – Код методу Update у скрипту GameManager.

Окрім цього методу в класі присутній два невеликих, один для отримання положення гравців на карті по координаті X (GetLeederCarX), і другий для звернення до всіх гравців і зупинки їх машин (StopGame).

3.4 Генерація треку

Та як трек в даній грі є випадковим і нескінченним то генеруватися він буде безпосередньо при грі. Даний тип створення мапи називається процедурної

генерацією, а саме автоматичним створенням будь-якого ігрового контенту за допомогою алгоритмів.

Сама дорога буде створюватися з вже готових шматків з усіма потрібними параметрами та компонентами, це буде робитися за допомогою системи префабів. Префаб це тип ресурсу, що дозволяє зберігати весь об'єкт з його частинами та властивостями. Це свого роду шаблон для швидкого створення копій потрібного об'єкту на сцені. Величезним плюсом префаба є те що змінюючи сам префаб всі зміни переходять і на його копії, це максимально прискорює роботу з великою кількістю однакових об'єктів, але при цьому є можливість змінювати об'єкти та компоненти в окремих копіях префабу.

Для створення алгоритму генерації треку потрібно подумки поділити його на частини. Всього трек має 2 смуги кожна з яких може бути одним з семи типів. Тип відрізка лінії є сукупністю його положення по висоті і те похилий він або прямий (різницю між ними показано на рис. 3.4.1).

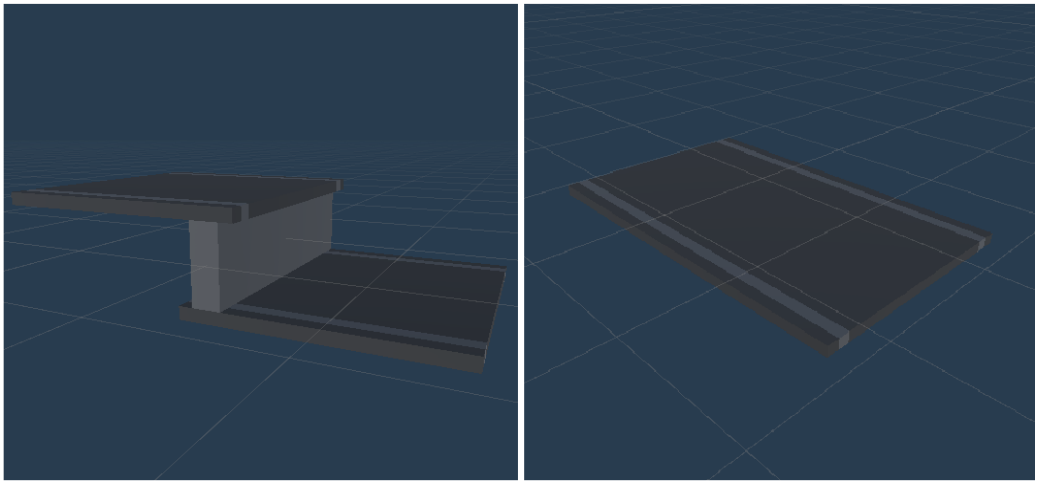


Рисунок 3.4.1 – Вигляд префабів лінії.

Для двигуна при створенні об'єкта потрібно вказати його майбутнє розташування в координатах X Y Z, поворот об'єкту і префаб за яким він буде створений, цю інформацію буде вийдуть з типу лінії і її положення на треку.

Префабів буде всього два: прямий і похилий. Різниця похилого і прямого префабу ще й в тому що похилий трек має компонент BoxCollider який є тригером,

у випадки зіткнення з ним параметр isDead в скрипті машини гравця стає в положення true. Цей колайдер встановлено так щоб машина не перетиналася з ним коли спускається.

Як писалося вище для коректної генерації нової ділянки смуги потрібно знати її розташування (права вона чи ліва), яка вона за рахунком на треку і якого типу була минула ділянка. Трек ніколи не змінює напрямок і завжди прагне на +X, з цього виходить що координати Z у його смуг статичні та у кожного свої. Щоб дізнатися координату X потрібно загальна кількість вже згенерованих ділянок треку додавши до нього 1 помножити на довжину відрізка. Координата Y, поворот і тип префабу це параметри за які відповідає тип лінії, він буде вибирається випадково зі списку можливих залежно від попереднього типу лінії. На базі цього був розроблений програмний код показаний на рис. 3.4.2 що знаходиться в класі Track Generator.

```
private void SpawnLine(int LineType, int position, float z)
{
    float y = 0;
    switch (LineType)
    {
        case 2:
            y = sigmentLengthY * 2 + sigmentStartY;
            break;
        case 12:
        case 11:
        case 0:
            y = sigmentLengthY * 1 + sigmentStartY;
            break;
        case 22:
        case 21:
        case 10:
            y = sigmentLengthY * 0 + sigmentStartY;
            break;
    }

    Vector3 linePos = new Vector3(sigmentLength * position, y, z);
    switch (LineType)
    {
        case 2:
        case 12:
        case 22:
            PhotonNetwork.InstantiateRoomObject(straightTrackPlefab.name, linePos, Quaternion.identity);
            break;
        case 0:
        case 10:
            PhotonNetwork.InstantiateRoomObject(bumpTrackPlefab.name, linePos, Quaternion.Euler(180, 0, 0));
            break;
        case 11:
        case 21:
            PhotonNetwork.InstantiateRoomObject(bumpTrackPlefab.name, linePos, Quaternion.identity);
            break;

    }
}
```

```
private void CellGenerate(int spawnCellCount)
{
    Debug.Log("TG: cellGen");
    int newf = 0;
    int news = 0;

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

        newf = LineTypeGen(lastFirstLineType);
        news = LineTypeGen(lastSecondLineType);
        SpawnTrackCell(newf, news);
        lastFirstLineType = newf;
        lastSecondLineType = news;
    }
}
```

```
private void SpawnTrackCell(int FirstLineType, int SecondLineType)
{
    SpawnLine(FirstLineType, cellCount, firstLinePosZ);
    SpawnLine(SecondLineType, cellCount, secondLinePosZ);
    cellCount++;
}
```

Рисунок 3.4.2 – Код, який відповідає за створення відрізків треку.

Список можливих це ті типи треку які можуть логічно бути після поточного, наприклад після підйому наступний тип повинен бути на рівень вище. Продумуючи цей список потрібно врахувати що оскільки рівнів висоти в грі замислюється 3 то

логічно що неприпустимо створювати підйом зверху й спуск знизу, також для балансу і меншою вимогливості від реакції гравця варто створювати тільки рівні ділянки відразу після похилих. Підсумкові умови генерації можна побачити в таблиці 3.3.1

Таблиця 3.4.1 – Умови генерації треку.

Якщо	То
пряма середня	будь-яка з середніх
пряма зверху	пряма зверху або спуск
пряма знизу	пряма знизу або підйом
верхній спуск або нижній підйом	пряма середня
середній підйом	пряма верхня
середній спуск	пряма нижня

Вважаючи це була зроблена id система з двозначних чисел де десятки це положення по висоті, а розряд одиниць визначає чи буде лінія спускатися, підійматися або прямою. Наприклад у прямій лінії по середині id буде дорівнювати 12. У коді за цей вибір відповідає метод LineTypeGen,

Перед генерацією за алгоритмом потрібно створити 3-4 ділянок рівної дороги з лініями на одному рівні, щоб гравцям було простіше зорієнтуватися на початку, а після можна почати генерацію за алгоритмом, за це відповіде метод SpawnStartTrackCell, він викликається при запуску скрипта.

Також важливо розуміти коли потрібно згенерувати нову ділянку дороги, в даному проєкті була використана генерація в міру проходження карти, а саме коли відстань між гравцями та останньої платформою стає менше ніж довжина восьми префабів лінії. Ця логіка обробляється в Update. Проїхані платформи знищуються за непотрібністю схожим чином, за допомогою встановленого на префаб скрипта SelfDestruction. Цей скрипт стежить за місцем розташування об'єкта GameManager (який своєю чергою знаходиться на позиції першого гравця), і якщо об'єкт на якому

прикріплений скрипт знаходиться на достатній відстані від GameManager, то цей об'єкт знищиться.

Для синхронізації карти між гравцями частини треку будуть створюватися за допомогою методу створення в самому PUN, а власником даних об'єктів буде вважатися сама сцена в якій запущений скрипт генерації. Створювати та знищувати такі об'єкти зможе тільки хост-клієнт.

3.5 Персонаж та управління

Префаб гравця(він зображений на рис. 3.5.1) складається зі скрипта PlayerControls для обробки його поведінки, двох моделей (коліс і основи машини), UI, ігровий камери й колайдера для прорахунку фізики. В скрипті гравця обробляється його стан і алгоритми пересування по карті.

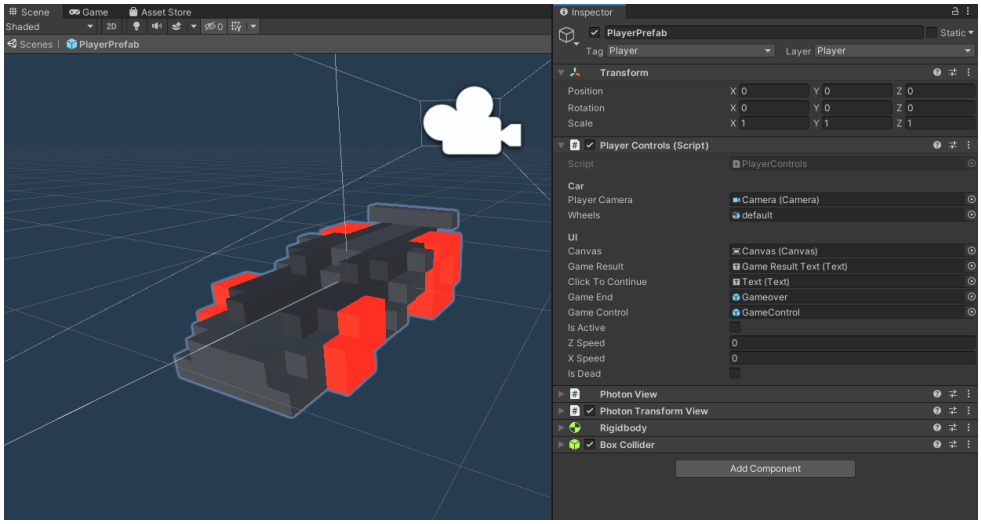


Рисунок 3.5.1 – Вигляд префабу персонажа і його компоненти.

Оскільки мультиплеєр в іграх це зазвичай дві синхронізація елементів у двох або більше копій ігор, то потрібно точно вказати який гравець управляє тими чи іншими об'єктами. Якщо, наприклад створити персонажів ігноруючи цей факт, то можна зіткнуться з низкою проблем таких як неможливість управляти ними або некоректно відображення ігрового світу. У PUN вже вбудована система володіння

об'єктами, так що все що потрібно від розробника це створити перевірки на володіння та коректне присвоєння[19].

Таким чином для коректного створення об'єкта гравця в ньому вимкнені всі компоненти крім моделі, колайдера та скрипта. В скрипті PlayerControls відбувається перевірка при появі об'єкта на сцені: у випадку якщо гравець є власником цього об'єкта, то для нього вмикаються відсутні компоненти, а також змінюється колір коліс машини. Зміна кольору зроблена для зручності розпізнавання машин.

Крім змін параметрів при створенні об'єкта скрипт PlayerControls також відповідає за управління, логіку машини та зміна її стану. Через те, що налаштування звуку і мови для кожного гравця свої то завантаження параметрів також відбувається в цьому скрипті.

Компоненти Photon View і Photon Transform View потрібні для синхронізації об'єкта між клієнтами. Компоненти Rigidbody і BoxCollider потрібні для обробки фізики: в той час як перший обробляє гравітацію об'єкта та інші його фізичні властивості то другий є свого роду межами тіла об'єкта. У префабі частин дороги також є компоненти Rigidbody і BoxCollider, з тією відмінністю що для дороги відключена будь-яка обробка фізики окрім зіткнення. Це дозволяє машинам їздити по дорозі та не провалюється крізь неї. Для того, щоб машині гравців не взаємодіяли між собою вони були виділені в окремий шар (layer) під назвою Player, а в налаштуваннях проєкту(рис 3.5.2) було зазначено що не потрібно обробляти фізику між об'єктами в цьому шарі.

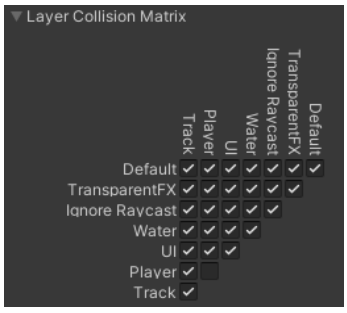


Рисунок 3.5.2 – Матриця колізії в налаштуваннях проєкту.

Також в компоненті Rigidbody була виключена обробка фізики обертання, це було зроблено оскільки обертання і переворот вже встановлені в коді PlayerControls, і відбуваються тільки під час виїзду з треку. Поворот був здійснено за допомогою корутини яка задає поворот машині кожен кадр, створюючи ефект плавного повороту. Корутини (Coroutine, співпрограми), це функції які працюють паралельно з основним процесом протягом деякого часу.

3.6 Інтерфейс

Кожен інтерфейс складається з об'єкта Canvas - це область, всередині якої знаходяться всі елементи UI (користувальницького інтерфейсу). Всі елементи UI повинні бути дочірніми цього Canvas[20]. Інтерфейс в грі присутній як в головному меню, так і в процесі поїздки

При запуску гри насамперед гравець потрапляє в головне меню, вид якого можна побачити на рисунку 3.6.1. За допомогою нього гравець буде шукати суперників і налаштовувати параметри гри.

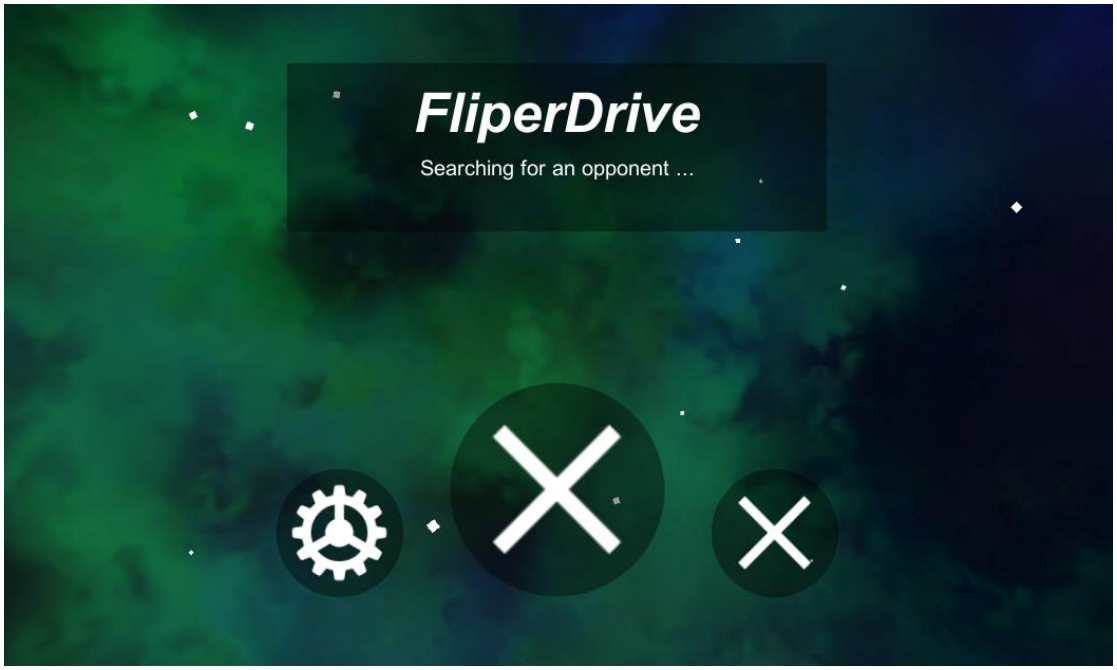


Рисунок 3.6.1 – Вигляд головного меню коли гравець шукає опонента.

У головному меню знаходиться напис з назвою гри, і три кнопки:

а) Кнопка налаштувань при натисканні якої з'являються підменю з можливістю налаштування параметрів гри, а саме звуку і мови.

б) Кнопка для пошуку суперника для гри. При пошуку змінює свою іконку на хрестик і стає кнопкою припинення пошуку.

в) Кнопка для виходу з програми.

Меню налаштувань(рис. 3.6.2) складається повзунка гучності музики та двох кнопок зміни мови, а також кнопки повернення в головне меню.

Рисунок 3.6.2 – Вигляд меню налаштувань.

В процесі гри, а саме при її завершенні гравцеві з'являються повідомлення про те що гра програна або виграна разом з пропозицією перейти на головне меню.

Все управління UI відбувається в класах PlayerControls та LobbyManager. Управління полягає у виведенні тексту в поля для відображення та зміна видимості певних елементів або цілих частин меню. Текст буде виведений на мові який вибрав гравець в головному меню. З доступних мов є англійська та українська.

Всі параметри пов'язані зі звуком і мовою зберігаються через систему PlayerPrefs, вона дозволяє зберігати інформацію між ігровими сесіями. можна зберігати рядки, цілі числа і числа з рухомою комою в реєстрі платформи користувача. Це метод зберігання інформації підходить для невеликих обсягів даних. На рис. 3.6.2 зображений код, який відповідає за завантаження параметрів, а в разі їх відсутності – привласнення значень за замовчуванням.

```
private void LoadSettingsOrSetDefault()
{
    if (!PlayerPrefs.HasKey(soundValuePPkey))
    {
        PlayerPrefs.SetFloat(soundValuePPkey, defaultSoundValue);
        AudioListener.volume = defaultSoundValue;
        SoundValue.value = defaultSoundValue;

    }
    else
    {
        AudioListener.volume = PlayerPrefs.GetFloat(soundValuePPkey);
        SoundValue.value = (PlayerPrefs.GetFloat(soundValuePPkey));
    }

    if (!PlayerPrefs.HasKey(languagePPkey))
    {
        PlayerPrefs.SetString(languagePPkey, defaultLanguage);

    }
    else
    {
        language = PlayerPrefs.GetString(languagePPkey);
    }
}
```

Рисунок 3.6.3 – Код методу LoadSettingsOrSetDefault у LobbyManager.

Завантаження параметрів повинно відбуватися кожного разу при завантаженні сцени, тому при запуску самих гонок в скрипті PlayerControls для кожного гравця окремо завантажуються його параметри. Після, дані отримані зі сховища беруть участь в відображенні користувацького інтерфейсу і встановленні гучності. Наприклад в коді (рис. 3.6.3) можна подивитися на метод StopCar в PlayerControls. Там є код який відповідає за виклик UI елемент підсумків гри.

```
public void StopCar()
{
    isActive = false;
    if (photonView.IsMine)
    {
        if (isDead)
        {
            gameResult.text = language == "EN" ? "DEFEAT!" : "ПОРАЗКА!";
            Debug.Log("Вы проиграли");

            gameControl.SetActive(false);
            gameEnd.SetActive(true);
        }
        else
        {

            gameResult.text = language == "EN" ? "VICTORY!" : "ПЕРЕМОГА!";
            Debug.Log("Вы ПОБЕДИЛИ!!!");

            gameControl.SetActive(false);
            gameEnd.SetActive(true);

        }
        clickToContinue.text = language == "EN" ? "Click to return to the menu." : "Натисніть щоб повернуться в меню.";
    }

}
```

Рисунок 3.6.4 – Метод StopCar.

3.7 Ефекти та музика

Візуальні ефекти та графіка були зведені до мінімуму для кращої швидкої роботи на дуже слабких пристроях і простоти картинки. З основних візуальних ефектів можна виділити обертається SkyBox у вигляді космічного простору і летючі куби. Обидва ефекти знаходяться в головному меню і безпосередньо у самій грі.

За обробку обертання SkyBox відповідають скрипти LobbyManager і GameLogicManager: кожен кадр відбувається зміна параметра повороту в налаштуваннях рендеру SkyBox таким чином, щоб створювався ефект плавного обертання.

Ефект летючих кубів створюється за допомогою об'єкта з компонентом генератора частинок (Particle system), він створює невеликі куби та рухає їх у випадковому напрямку, живуть ці об'єкти кілька десятків секунд і в залежності їх віку змінюють свій розмір таким чином, щоб це було схоже на плавну появу і

зникнення. Всього одночасно об'єкт буде створювати 50 таких частинок. На сцені головного меню даний об'єкт статично стоїть в полі видимості гравця, а на треку прикріплений до об'єкта GameManager як дочірній, оскільки він пересувається слідом за гравцями. Таким чином гравці будуть завжди знаходиться в центрі поля дії генератора частинок і ніколи не проїдуть його. На рис. 3.7.1 зображений цей ефект.

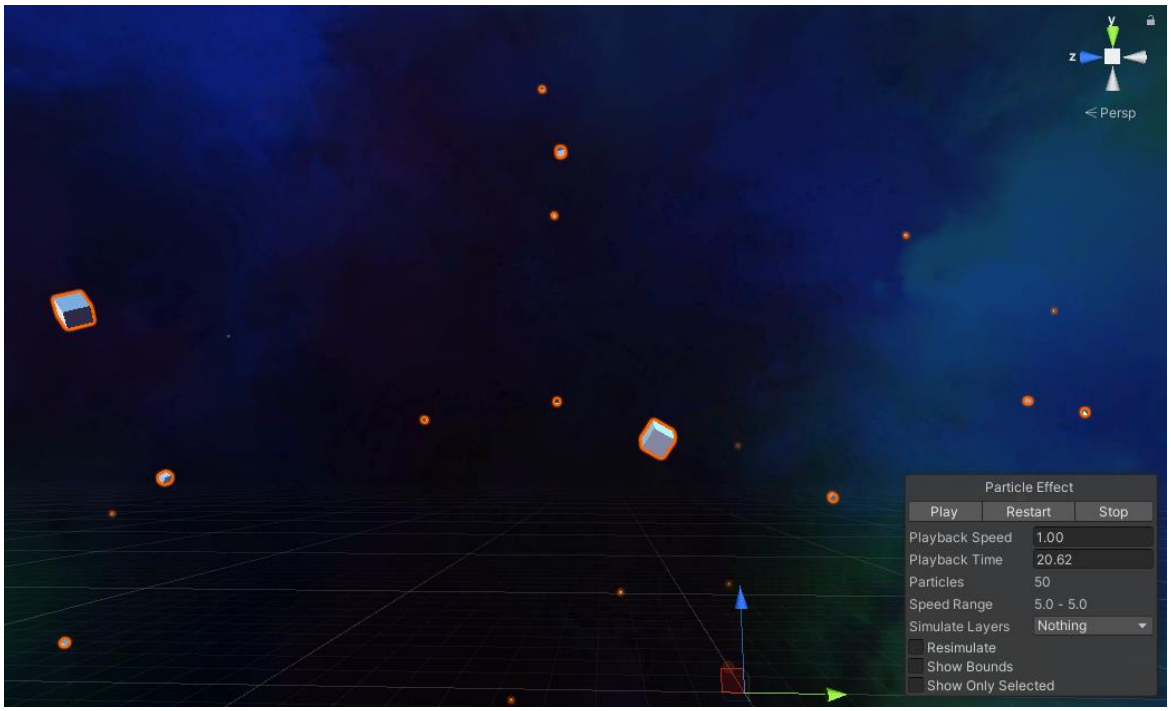


Рисунок 3.7.1 – Вигляд летючих кубів у редакторі.

Музика в грі є досить важливою частиною, її мета частіше за все задати настрій в процесі гри або оповістити гравця про щось. В грі у якості музики використовується невеликий зациклений аудіотрек, щоб задати настрій на поїздку.

4 ВИСНОВКИ

Підбиваючи підсумки проведеного аналізу можна сказати що ринок ігрової індустрії буде ще довго зростати. Починаючи з самих джерел в 1970-х роках закінчуючи нашими днями можна побачити як майже за півстоліття аудиторія відеоігор зросла майже з нуля до чверті населення землі. Зараз, завдяки доступності інтернету і мобільних пристроїв ця індустрія переживає один з кращих періодів: аудиторія зростає і розширюється завдяки спрощенню доступу до ігор. За останні десять років поріг входження значно знизився, на додаток до гравців-любителів, які цілеспрямовано купували ігри на ПК і консолі, з'явилася категорія користувачів які просто хочуть невимушено скоротати небагато часу, наприклад по дорозі додому з роботи. Це призвело до створення нової та досить великий ніші мобільних відеоігор на ринку. Як правило, ті хто грали в дитинстві продовжують грати й зараз та знайомлять з цим вже своїх дітей. Також, вже не популярна думка що ігри це заняття суто для хлопців. В наш час відеоігри це для всіх, у кого є на чому в них грати.

Разом з самими відеоіграми розвиваються й інструменти для їх розробки. За підсумком порівняння інструментів не можна назвати безумовно кращий продукт який підійшов би для всіх рішень. Як і самі ігри намагаються бути не схожими один на одного так і інструменти для їх створення відрізняється цілями та можливостями. В ході проведеного дослідження було обрано ігровий двигун Unity, тому, що був оптимальним і доступним рішенням для поставленого завдання.

Під час роботи над дипломною роботою був створений програмний продукт у вигляді гри жанру аркадних гонок з можливість грати через мережу. У майбутньому даний проєкт можна розвинути й доопрацювати: поліпшивши графіку або урізноманітнити ігровий процес. Також є можливість швидко перенести даний продукт на інші платформи, що підвищить охоплення аудиторії.

5 ПЕРЕЛІК ПОСИЛАНЬ

1.Вплив COVID-19 та карантинних обмежень на економіку України. Центр прикладних досліджень, 2020. Режим доступу: https://www.kas.de/documents/270026/8703904

2. The History and Evolution of the Video Games Market [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://www.businessinsider.com/the-history-and-evolution-of-the-video-games-market-2017-1 (дата звернення 29.04.2021). – Назва з екрана.

3.Епідемія коронавірусу: продажі комп'ютерних ігор різко виросли / НВ [Електронний ресурс]: [Інтернет-портал]. – Режим доступу: https://nv.ua/techno/games/epidemiya-koronavirusa-prodazhi-kompyuternyh-igr-rezko-vyrosli-50080250.html (дата звернення 29.04.2021). – Назва з екрана.

4. Етапи розробки комп’ютерних ігор – FUGAS [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://blog.fugas.space/gamedev-stages/ (дата звернення 29.04.2021). – Назва з екрана.

5. Гейм-дизайнер – Proforientator.info [Електронний ресурс]: [Веб-сайт]. – Режим доступу: http://proforientator.info/?page_id=3409 (дата звернення 29.04.2021). – Назва з екрана.

6. How Unity3D Became a Game-Development Beast [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/ (дата звернення 29.04.2021). – Назва з екрана.

7. Unity 2.0 game engine now available [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://www.macworld.com/article/187693/unity-18.html (дата звернення 29.04.2021). – Назва з екрана

8. Unity 3 brings very expensive dev tools at a very low price | Ars Technica [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://arstechnica.com/information-technology/2010/09/unity-3-brings-very-expensive-dev-tools-at-a-very-low-price/?comments=1 (дата звернення 29.04.2021). – Назва з екрана.
9. Unity Releases 2D Tools with 4.3 Update | Unity [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://unity.com/our-company/newsroom/unity-releases-2d-tools-4-3-update (дата звернення 29.04.2021). – Назва з екрана.
10. What's new in Unity 5.0 – Unity [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://unity3d.com/unity/whats-new/unity-5.0 (дата звернення 29.04.2021). – Назва з екрана.
11. What's new in Unity 2017.1 – Unity [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://unity3d.com/unity/whats-new/unity-2017.1.0 (дата звернення 29.04.2021). – Назва з екрана.
12. Get Unity 2019.3 here! | Unity [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://unity.com/releases/2019-3 (дата звернення 29.04.2021). – Назва з екрана.
13. Bolt visual scripting is now included in all Unity plans | Unity Blog [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://blog.unity.com/news/bolt-visual-scripting-is-now-included-in-all-unity-plans (дата звернення 29.04.2021). – Назва з екрана.
14. 10 Reasons to Use Godot Engine for Developing Your Next Game [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://www.makeuseof.com/tag/reasons-godot-engine-game-development/ (дата звернення 29.04.2021). – Назва з екрана.
15. Unity vs Cryengine: 9 Point super comparison – VionixStudio [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://vionixstudio.com/2020/01/17/unity-vs-cryengine-game-engine-comparison/ (дата звернення 29.04.2021). – Назва з екрана

16. Source Engine 2 doesn't have hidden costs or royalties - but is Steam exclusive - VG247 [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://www.vg247.com/2015/03/05/source-engine-2-doesnt-have-hidden-costs-or-royalties-but-is-steam-exclusive/ (дата звернення 29.04.2021). – Назва з екрана.
17. Photon PUN Pricing Plans | Photon Engine [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://www.photonengine.com/PUN/Pricing (дата звернення 29.04.2021). – Назва з екрана.
18. Unity - Manual: Unity User Manual 2020.3 (LTS) [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://docs.unity3d.com/Manual/index.html (дата звернення 29.04.2021). – Назва з екрана.
19. Ownership & Control | Photon Engine [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://doc.photonengine.com/en-US/pun/current/gameplay/ownershipandcontrol (дата звернення 29.04.2021). – Назва з екрана.
20. Canvas | Unity UI | 1.0.0 l [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/UICanvas.html (дата звернення 29.04.2021). – Назва з екрана.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

РОЗРОБКА БАГАТОКОРИСТУВАЦЬКОЇ ГРИ В ЖАНРІ АРКАДНОЇ ГОНКИ МОВОЮ C#

Виконала студентка 5 курсу
Групи ППЗ-51
Прежина А.А.
Керівник роботи
Шевченко С.М.

Київ – 2021

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи:** здійснення аналізу сучасних інструментальних засобів для побудови відеоігор.
- **Об'єкт дослідження:** Розробка відеоігор.
- **Предмет дослідження:** Багатокористувацькі ігри.

АНАЛОГИ

Distance

Traffic Racer

Asphalt 9

АНАЛОГИ

Порівняння продукту з аналогами

Назва	Характеристики		
	3D графіка	Багатокористувацький режим	Процедурна генерація дороги у грі
FliperDrive (Цей проєкт)	Так	Так	Так
Asphalt 9	Так	Так	Ні
Earn to Die 2	Ні	Ні	Ні
Distance	Так	Так	Ні
Traffic Racer	Так	Ні	Так

ТЕХНІЧНІ ЗАВДАННЯ

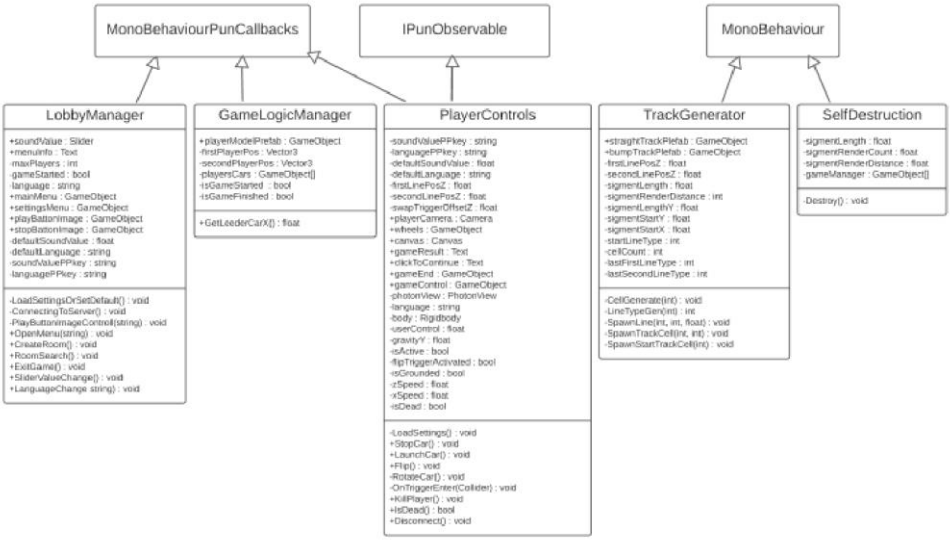
- *Провести дослідження на тему розробки комп'ютерних ігор*
- *На базі проведених досліджень спроєктувати продукт*
- *Створення багатокористувацької частини*
- *Створення машини та опрацювання її поведінки*
- *Створення алгоритму генерація треку*
- *Створення призначеного для користувача інтерфейсу*

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Для реалізації завдання використовувався ігровий двигун Unity, що дозволяє створювати 2D і 3D ігри під більшість популярних платформ.

Для реалізації багатокористувацької частини у грі використовувався Photon Unity Networking, це бібліотека для створення з'єднання між гравцями для ігор на двигуну Unity.

МЕТОДИ ТА КЛАСИ ПРОГРАМИ



МЕТОДИ ТА КЛАСИ ПРОГРАМИ

```
void Update()
{
    RenderSettings.skybox.SetFloat("_Rotation", Time.time * 0.4f);
    if (!isGameStarted)
        {
        playersCars = GameObject.FindGameObjectsWithTag("Player");
        if ( PhotonNetwork.PlayerList.Length == playersCars.Length)
            {
                // starting when all players load ////////////////////////////////////////////
                isGameStarted = true;
                foreach (var obj in playersCars)
                {
                    obj.GetComponent<PlayerControls>().LaunchCar();
                }
                ////////////////////////////////////////////////////////////////////////////////
            }

        }
        else
        {
            if (!isGameFinished)
            {
                transform.position = new Vector3(GetLeederCarX(), 0, 0);
                foreach (var obj in playersCars)
                {
                if (obj.GetComponent<PlayerControls>().IsDead()) { StopGame(); }
                    else
                    {
                        // if all players alive ////////////////////////////////////////////
                        if (obj.transform.position.x > transform.position.x) transform.position = new Vector3(GetLeederCarX(), 0, 0);
                        ////////////////////////////////////////////////////////////////////
                    }
                }
            }
        }
}
```

Метод Update у GameLogicManager, він викликається один раз за кадр після запуску скрипта і протягом всієї його активності.

МЕТОДИ ТА КЛАСИ ПРОГРАМИ

```
private int LineTypeGen( int lastLinetype)
{
    int result = 99;
    int rand = (int)Random.Range(0, 30);

    switch (lastLinetype)
    {

        case 12:
            if (rand == 0){ result = 10;}
            else if (rand == 1){ result = 11;}
            else if (rand <= 2){result = 12;}
            break;

        case 2:
            if (rand == 0){result = 0;}
            else{result = 2;}
            break;

        case 22:
            if (rand == 0){result = 21;}
            else{result = 22;}
            break;
        case 0:
        case 21:
            result = 12;
            break;
        case 11:
            result = 2;
            break;
        case 10:
            result = 22;
            break;
    }
    return result;
}
```

```
private void SpawnLine(int LineType, int position, float z)
{
    float y = 0;
    switch (LineType)
    {
        case 2:
            y = sigmentLengthY * 2 + sigmentStartY;
            break;
        case 12:
        case 11:
        case 1:
            y = sigmentLengthY * 1 + sigmentStartY;
            break;
        case 22:
        case 21:
        case 10:
            y = sigmentLengthY * 0 + sigmentStartY;
            break;
    }

    Vector3 linePos = new Vector3(sigmentLength * position, y, z);
    switch (LineType)
    {
        case 2:
        case 12:
        case 22:
            PhotonNetwork.InstantiateRoomObject(straightTrackPlefab.name, linePos, Quaternion.identity);
            break;
        case 0:
        case 10:
            PhotonNetwork.InstantiateRoomObject(bumpTrackPlefab.name, linePos, Quaternion.Euler(180, 0, 0));
            break;
        case 11:
        case 21:
            PhotonNetwork.InstantiateRoomObject(bumpTrackPlefab.name, linePos, Quaternion.identity);
            break;

    }
}
```

Методи LineTypeGen і SpawnLine у TrackGenerator, що відповідають за вибір типу для наступного відрізка смуги і його створення на сцені відповідно.

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. ПРЕЖИНА А.А. РОЗРОБКА БАГАТОКОРИСТУВАЦЬКОЇ ГРИ В ЖАНРІ АРКАДНОЇ ГОНКИ. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інфокомунікаційних технологіях». Збірник тез. – К.: ДУТ, 2021. – С. 108-109
   - http://www.dut.edu.ua/uploads/n_9058_51926054.pdf

ВИСНОВКИ

У процесі роботи було проведено:

1. Аналіз явища комп'ютерних ігор і його актуальність;
2. Аналіз інструментів для розробки;
3. Розробка гри в жанрі аркадних гонок.

ДЯКУЮ ЗА УВАГУ!