МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

Перший (бакалаврський)
(рівень вищої освіти)

на тему **Розробка ігрового застосунку з використанням оптимального руху об`єктів на основі методів штучного інтелекту**

Виконав: студент 4 курсу, групи 6.1219-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
Д.О. Величко
(ініціали та прізвище)
Керівник доцент, к.ф.-м.н., доцент І.А.Скрипник
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дискус»
Р.О.Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електроніки, інформаційних систем та програмного забезпечення

Рівень вищої освіти _____перший (бакалаврський)_____

Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)

Освітня програма ___Програмне забезпечення систем
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Т.В. Критська_
“_01_” ___березня__ 2023_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Величку Данилу Олександровичу
(прізвище, ім’я, по батькові)

1.Тема роботи __Розробка ігрового застосунку з використанням оптимального руху об`єктів на основі методів штучного інтелекту

керівник роботи Скрипник Ірина Анатоліївна, к.ф.-м.н., доцент,_____
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від 29.12.2022 р. № 1893-с

2. Строк подання студентом кваліфікаційної роботи__14.06.2023_____

3. Вихідні дані бакалаврської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження проблеми розробки ігор, особливості розробки в Unity;
- створення програмного продукту та його опис;
- дослідження поставленої проблеми та розробка висновків.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

_____слайдів презентації

6. Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.03.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Аналіз предметної області	22.04.23	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	25.04.23	виконано
3	Аналіз існуючих методів рішення	26.04-30.04.23	виконано
4	Аналіз методів розробки ігор на Unity	01.05-04.05.23	виконано
5	Аналіз засобів розробки ігор на Unity	05.05-08.05.23	виконано
6	Узгодження подальших дій з науковим керівником	10.05.23	виконано
7	Проектування гри	12.05.23	виконано
8	Програмна реалізація застосунку	14.05-22.05.23	виконано
9	Представлення отриманих результатів науковому керівнику та узгодження плану подальшого дослідження	25.05.23	виконано
10	Реалізація ШІ як суперника у грі	26.05-03.06.23	виконано
11	Тестування застосунку та можливостей ШІ	05.06.23	виконано
12	Оформлення звіту	8.06-11.06.23	виконано
13	Оформлення презентації.	12.06-15.06.23	виконано

Студент _____ Д.О.Величко
(підпис) (прізвище та ініціали)

Керівник роботи _____ І.А.Скрипник
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ І.А. Скрипник
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок – 68

Рисунків – 26

Джерел – 11

Величко Д.О. Розробка ігрового застосунку з використанням оптимального руху об`єктів на основі методів штучного інтелекту: кваліфікаційна робота бакалавра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник доцент Скрипник І.А. Запоріжжя : ЗНУ, 223. 68с.

Кваліфікаційна робота присвячена вивченню можливостей розробки ігрових застосунків на Unity та імплементуванню ШІ у якості суперника. У роботі було проведено аналіз існуючих технологій розробки ігор та методів імплементування штучного інтелекту. Були досліджені методи написання скриптів на C# для надання ШІ можливості реагувати на навколишнє середовище.

Для реалізації застосунку було використано інструменти рушія Unity та мову програмування C#. Були розроблені скрипти для орієнтування ШІ на трасі та мапи для гри за допомогою редактора мап Unity.

У роботі було показано, що Unity є багатофункціональним та при цьому зручним рушієм для розробки ігрових застосунків. Задані штучному інтелекту скрипти та методи навігації дозволяють йому орієнтуватися на трасі та конкурувати з гравцем. Результати тестів підтверджують високу якість роботи ігрового застосунку при низьких системних вимогах та відображають можливості розвитку штучного інтелекту.

Кваліфікаційна робота містить докладний опис процесу створення ігрового застосунку, дослідження існуючих рішень та їх переваг або недоліків, результати тестування та аналіз отриманих результатів. Робота є внеском у розвиток науки та технологій використання ШІ у ігрових застосунках, та при подальшій розробці може стати конкурентоспроможним продуктом у світі ігор.

Ключові слова: штучний інтелект, Unity, скрипт, моделювання трас, системні вимоги.

ANNOTATION

Pages – 68

Drawings – 26

Sources – 11

Velychko D.O. Development of a game application using the optimal movement of objects based on artificial intelligence methods: bachelor's thesis in specialty 121 "Software Engineering" / scientific adviser, associate professor Skrypnyk I.A. Zaporizhzhia: ZNU, 223. 68p.

The qualification work is devoted to the study of the possibilities of developing game applications on Unity and the implementation of AI as a competitor. The work analyzes existing game development technologies and methods of implementing artificial intelligence. Methods of writing C# scripts to enable AI to respond to the environment were investigated.

The tools of the Unity engine and the C# programming language were used to implement the application. Scripts were developed to guide the AI on the track and maps for the game using the Unity map editor.

The paper shows that Unity is a multifunctional and convenient engine for developing gaming applications. The scripts and navigation methods defined for the AI allow it to navigate the track and compete with the player. The test results confirm the high quality of the gaming application with low system requirements and reflect the possibilities of artificial intelligence development.

The qualification work contains a detailed description of the process of creating a gaming application, a study of existing solutions and their advantages or disadvantages, test results, and an analysis of the results. The work is a contribution to the development of science and technology of using AI in gaming applications, and with further development can become a competitive product in the world of gaming.

Keywords: artificial intelligence, Unity, script, track modeling, system requirements.

ЗМІСТ

ВСТУП

Актуальність теми

Штучний інтелект (ШІ) в останні роки став предметом інтенсивних досліджень та застосувань у різних галузях, включаючи геймдевелопмент. Завдяки постійному розвитку технологій, штучний інтелект стає все більш ефективним у вирішенні складних завдань і внесенні нових можливостей у гейммплей.

Однією з актуальних областей використання штучного інтелекту в гральній індустрії є пошук найкращого шляху на гоночній трасі. Це важливий аспект реалістичних гоночних ігор, який впливає на досвід гравців та рівень виклику. Створення імітації реального водія, який шукає оптимальну лінію руху на трасі, є складним завданням, яке вимагає високої обчислювальної потужності та складних алгоритмів.

Одним із підходів до пошуку найкращого шляху є використання слідування невидимим маркерам, розташованим вздовж гоночної траси. Ці маркери вказують штучному інтелекту оптимальну лінію руху та допомагають уникнути перешкод і небажаних маневрів. ШІ аналізує позицію та відстань до маркерів, використовує алгоритми навігації та приймає рішення про найкращий курс дій.

Гра на Unity з використанням штучного інтелекту, який шукає найкращий шлях на гоночній трасі за допомогою слідування невидимим маркерам, має кілька переваг. По-перше, це створює більш інтелектуальних та вимогливих опонентів для гравців, що підвищує рівень виклику та інтересу. По-друге, це дозволяє створювати різноманітні гоночні траси з різними рівнями складності, де кожна траса може мати власну унікальну лінію руху для штучного інтелекту. Це забезпечує різноманіття та непередбачуваність геймплею, підвищуючи переігрованість гри.

Використання штучного інтелекту для пошуку найкращого шляху на гоночній трасі за принципом слідування невидимим маркерам у грі на Unity є

актуальним та цікавим напрямком в геймдевелопменті. Це дозволяє створювати гри зі складним інтелектом опонентів, варіативними гоночними трасами та захоплюючим геймплеєм, що підвищує якість інтерактивного досвіду гравців. З розвитком технологій ШІ, можна очікувати ще більш реалістичних та захоплюючих гоночних ігор у майбутньому.

Мета дослідження

Розробка гри з використанням ШІ, який знаходить оптимальний шлях за допомогою невидимих маркерів.

Завдання дослідження

Розглянути особливості та існуючі рішення (аналоги) для використання ШІ. Проаналізувати існуючі підходи та технології до розробки ігор та вибрати відповідну для створення гри з використанням ШІ.

Об’єкт дослідження

Об’єктом дослідження є процес розробки гри та імплементування ШІ.

Предмет дослідження

Предметом дослідження є технології створення власної гри, в якій противником виступає ШІ та розгляд його можливостей з обраним методом орієнтування на трасі.

Методи дослідження

Теоретичні – обробка літературних джерел, синтез дослідження та аналіз досліджуваного матеріалу.

Практичне значення одержаних результатів

Практичне значення одержаних результатів дослідження полягає у тому, що розробка власної гри з використанням ШІ підвищує її цінність та інтерес для користувача.

Глосарій

Unity. Інтегроване середовище розробки (IDE), яке використовується для створення і редагування графічних ігор. Unity є потужним інструментом для розробки ігор на різних платформах.

Графічний двигун. Програмне забезпечення, яке відповідає за обробку графіки, фізики та інших візуальних ефектів в грі. Unity має вбудований графічний двигун, який дозволяє створювати реалістичні графічні ефекти.

Сцена. Простір, в якому розгортається гра. Сцена включає в себе об'єкти, текстури, освітлення та інші компоненти, які утворюють візуальне середовище гри.

Об'єкт. Основний елемент гри, який може мати свою графіку, поведінку та інші характеристики. Об'єкти в Unity можуть бути гравцями, ворогами, об'єктами оточення тощо.

Компонент. Частина об'єкта, яка визначає його властивості і поведінку. Компоненти можуть бути додані до об'єктів для надання їм різних функцій, таких як рух, фізика, звук, колізії та інші.

Скрипт. Файл програмного коду, який містить інструкції для керування поведінкою об'єктів в грі. У Unity використовується мова програмування C# для написання скриптів.

Анімація. Процес створення руху і динаміки об'єктів у грі. Unity підтримує систему анімації, яка дозволяє створювати різні типи анімацій, такі як рух персонажів, зміна положення об'єктів, ефекти тощо.

Фізика: Взаємодія об'єктів у грі згідно з фізичними законами. Unity має вбудовану систему фізики, яка дозволяє симулювати реалістичні фізичні ефекти, такі як гравітація, колізії, реакції на удари тощо.

UI (користувацький інтерфейс). Елементи інтерфейсу, які відображаються на екрані гри і взаємодіють з гравцем. UI включає в себе кнопки, текстові поля, меню, панелі тощо.

Оптимізація. Процес покращення продуктивності і швидкості роботи гри. Оптимізація включає в себе редагування коду, видалення непотрібних ресурсів, оптимізацію графіки та інші заходи для забезпечення плавного ігрового процесу.

Штучний інтелект (ШІ). Галузь комп'ютерних наук, яка займається розробкою програм та алгоритмів, що дозволяють комп'ютеру виконувати завдання, що зазвичай потребують інтелекту людини.

Оптимальний шлях. Шлях, який максимізує ефективність, швидкість та успішність переміщення гравця на гоночній трасі.

Гоночна гра. Відеогра, в якій гравці змагаються в гонках на гоночних трасах, спробуючи подолати опонентів і стати переможцем.

Слідування маркерам. Техніка, за допомогою якої штучний інтелект визначає оптимальний шлях руху, керуючись невидимими маркерами, розташованими вздовж траси.

Маркери. Позначення або об'єкти, розташовані на гоночній трасі, які слугують покажчиками для штучного інтелекту. Вони вказують оптимальну траєкторію руху гравця.

Геодезичні координати. Система координат, що використовується для визначення точного положення маркерів та гравця на гоночній трасі.

Алгоритм навігації. Алгоритм, який допомагає штучному інтелекту приймати рішення щодо вибору найкращого шляху, враховуючи дані про маркери, положення та умови гри.

Рухова стратегія. План дій, який штучний інтелект використовує для керування рухом гравця на трасі, з урахуванням маркерів та умов гри.

Динамічні умови. Змінні фактори, які можуть вплинути на рух гравця, такі як погода, стани дороги, перешкоди або зміна швидкості транспортного засобу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд літературних джерел

Доступна література пропонує вивчення таких аспектів ШІ, як навігація, поведінкові агенти, прийняття рішень та машинне навчання [3]. Вони надають практичні приклади, що допоможуть читачам зрозуміти концепції і застосувати їх у власних проектах [1]. Завдяки цим джерелам, розробники зможуть створити гру, де інтелектуальні персонажі будуть шукати найкращий шлях на гоночній трасі, слідуючи невидимим маркерам.

Також розглядаються різні алгоритми та методи, що допоможуть досягти більшої реалістичності інтелектуальних персонажів у грі. Вони пропонують широкий спектр завдань та рішень, пов'язаних з розробкою інтелектуальних персонажів у грі, та надають можливість засвоїти різноманітні техніки та підходи до розробки ШІ в Unity [2].

Розробка ігор на Unity зазвичай зустрічається з кількома викликами. Перш за все, це складність процесу розробки. Особливо для одиночних розробників або невеликих команд, створення ефективно працюючого застосунку потребує значних зусиль, часу, знань C# і технічної експертизи.

Друга проблема пов'язана з вимогами до продуктивності. Щоб забезпечити плавний геймплей і уникнути затримок, ігровий застосунок повинен бути оптимізованим. Це може вимагати великих інвестицій у комп'ютерну техніку і програмне забезпечення, а також освіти з оптимізації коду.

Нарешті, нестабільність ринку ігор є ще одним викликом. Розробники ігрових застосунків повинні забезпечити зацікавленість гравців до свого продукту, щоб відповідати змінам і потребам ринку, який є непередбачуваним.

Останнім питанням залишається імплементування ШІ у ігровий застосунок, на сьогоднішній день існує багато засобів для створення ШІ у якості суперника, наприклад машинне навчання або скінченні автомати з заданими

скриптами на мові C# які надають їм можливість змінювати свій стан та реагувати на зміни в оточуючому середовищі.

1.2 Дослідження розробки гоночних ігор з використанням ШІ

Гоночні ігри завжди були популярним жанром серед геймерів. Розробники постійно прагнуть покращити реалізм та інтенсивність гоночного досвіду для гравців. Штучний інтелект (ШІ) став важливим інструментом у реалізації цих цілей. Використання ШІ у гоночних іграх дозволяє створювати вражаючий геймплей, реалістичних опонентів та захоплюючі траси, де гравці можуть випробувати свої навички у керуванні транспортним засобом. У цій статті ми дослідимо перспективи використання ШІ у розробці гоночних ігор та розглянемо деякі цікаві тенденції та підходи.

Одним з ключових аспектів гоночних ігор є реалістична навігація інтелектуальних опонентів по трасі. Використання ШІ дозволяє створювати опонентів, які вміють ефективно шукати найкращий шлях на гоночній трасі. Це можливо завдяки принципу слідування невидимим маркерам, розташованим вздовж траси. Інтелектуальні опоненти можуть виявляти та аналізувати ці маркери, щоб визначити оптимальний шлях руху та приймати відповідні рішення щодо керування.

Ще одним важливим аспектом гоночних ігор є реалістичність поведінкових моделей інтелектуальних опонентів. Використання ШІ дозволяє розробникам створювати опонентів зі складними алгоритмами прийняття рішень, що робить їх більш схожими на реальних гравців. Інтелектуальні опоненти можуть виконувати тактичні маневри, виявляти стратегічну поведінку та реагувати на дії гравця. Це створює більш глибокий та викликаючий геймплей для гравців.

Unity є однією з провідних платформ для розробки гоночних ігор і використання штучного інтелекту. У недавні роки в Unity з'явилися передові інструменти та бібліотеки для розробки ШІ. Наприклад, система навігації

NavMesh дозволяє створювати реалістичну навігацію для інтелектуальних опонентів на гоночних трасах. Крім того, Unity ML-Agents Toolkit надає можливості для тренування інтелектуальних агентів за допомогою машинного навчання.

Рис.*1 Unity ML-Agents Toolkit*

З врахуванням швидкого розвитку штучного інтелекту та його застосування у гральній індустрії, можна очікувати, що використання ШІ у розробці гоночних ігор буде продовжувати зростати. Дослідники та розробники працюватимуть над вдосконаленням алгоритмів навігації, поведінкових моделей та інших аспектів ШІ для досягнення ще більшого рівня реалізму та викликів у гральних іграх.

Використання штучного інтелекту у розробці гоночних ігор на платформі Unity відкриває широкі можливості для створення захоплюючого та реалістичного геймплею. ШІ дозволяє реалізувати навігацію опонентів на гоночних трасах, реалістичні поведінкові моделі та забезпечує більш глибокий та викликаючий досвід для гравців. З розвитком передових методів штучного інтелекту в Unity, майбутнє використання ШІ у гоночних іграх обіцяє бути ще більш захоплюючим та реалістичним.

1.3 Аналіз існуючих гонок з використанням ШІ у якості суперника

Штучний інтелект (ШІ) поступово завойовує своє місце у світі комп'ютерних ігор, дозволяючи створювати реалістичний та викликаючий геймплей. Однією з захоплюючих областей є гонки, де ШІ виступає в ролі суперника гравцеві.

Рис.*2 Need for Speed: Rivals*

Перша гра, яку розглянемо, — "Need for Speed: Rivals". У цій грі ШІ виконує роль поліцейських, які полюють на гравця-гонщика. ШІ використовує складні алгоритми для прийняття рішень, включаючи вибір оптимального шляху переслідування та використання різних тактик для затримання гравця. Розглянемо плюси та мінуси даної гри на ринку ігрових застосунків з використанням ШІ у якості суперника:

Реалістична поведінка суперників: Завдяки використанню ШІ, суперники в грі Need for Speed Rivals проявляють високу рівень інтелекту та адап-

тивності. Вони здатні приймати стратегічні рішення, виявляти тактичну гнучкість та адекватно реагувати на дії гравця. Це створює реалістичне відчуття змагання зі справжніми суперниками.

Динамічність геймплею: ШІ суперників дозволяє створювати динамічну та непередбачувану гру. Суперники можуть змінювати свою стратегію, варіювати швидкість, маршрути та стилі ведення гонки, що робить кожну гонку унікальною та цікавою для гравця.

Виклик та навички: Гра проти ШІ суперників вимагає від гравця вміння приймати рішення на основі реального часу та адаптуватися до змінюючихся умов гонки. Це дозволяє розвивати навички гравця, сприяє покращенню реакційних здібностей та стратегічного мислення.

Мінуси гонки Need for Speed Rivals з використанням ШІ у якості суперника:

Прогнозованість: ШІ в грі може мати встановлені алгоритми та обмежену кількість реакційних шаблонів. Це може призводити до прогнозованості поведінки суперників та зменшення виклику для досвідчених гравців.

Відсутність емоційного аспекту: ШІ суперників може бути позбавлено емоційного спектру та виразності, що зменшує іммерсивність гри. Відсутність емоційного взаємодії з суперниками може знизити загальне задоволення від геймплею.

Потенційні помилки алгоритмів: У разі наявності недосконалих алгоритмів, ШІ суперників може допускати помилки та неправильні рішення, що може порушити баланс гри та створити ситуації, які виглядають неадекватними або невірогідними.

Висока вартість продукту та високі системні вимоги: через такі високі вимоги до системи та ціну далеко не всі зацікавлені гравці зможуть у повному масштабі оцінити як графічні так і функціональні можливості гри.

Враховуючи плюси та мінуси гонки Need for Speed Rivals з використанням ШІ в якості суперника, не всі гравці можуть насолоджуватись реалістичною грою з високим рівнем виклику та інтелектуального противника, проте

слід враховувати можливі обмеження та прогнозованість, які можуть виникнути через використання ШІ.

Рис.*3 Project CARS 2*

Іншим прикладом є гра "Project CARS 2", яка пропонує реалістичний досвід гоночних змагань. У цій грі ШІ виступає в ролі інтелектуальних гоночних автомобілів, які змагаються з гравцем. ШІ використовує різні стратегії та тактики, враховуючи характеристики траси та поведінку гравця, щоб досягти перемоги.

Розглянемо переваги та недоліки даного ігрового застосунку:

Переваги гонки Project CARS 2 з використанням ШІ у якості суперника:

Реалістична модель фізики: Гра Project CARS 2 відзначається високоякісною моделлю фізики, що створює реалістичне відтворення руху транспортних засобів. Застосування ШІ у якості суперників дозволяє їм дотримуватися правил дорожнього руху, реагувати на зміни умов дороги та виконувати складні маневри, що покращує реалістичність гри.

Адаптивність та інтелект суперників: ШІ суперників у Project CARS 2 може адаптуватися до стилю гри гравця та здійснювати адекватні дії залежно

від ситуації на трасі. Вони можуть реагувати на зміну погодних умов, травмуватися та впливати на стратегію гонки, що створює високий рівень виклику для гравця.

Різноманітність суперників: У грі Project CARS 2 ШІ суперників може мати різні рівні складності, стилі ведення гонки та стратегії. Це робить кожну гонку унікальною та непередбачуваною, дозволяючи гравцю отримати різноманітні виклики та емоції.

Недоліки гонки Project CARS 2 з використанням ШІ у якості суперника:

Стабільність та оптимізація: В деяких випадках можуть виникати проблеми зі стабільністю гри та її оптимізацією при використанні ШІ суперників. Це може призводити до зниження кадрової частоти, артефактів або затримок у відтворенні гри, що впливає на загальний досвід гравця.

Потенційні помилки алгоритмів: Використання ШІ у грі може супроводжуватися наявністю помилок у роботі алгоритмів, що керують поведінкою суперників. Це може призводити до ситуацій, коли суперники вчиняють неадекватні дії або несподівано поводяться, що може негативно вплинути на геймплей.

Однотонність стилю гри суперників: ШІ суперників у Project CARS 2 може виявляти певну однотонність в стилі гри та стратегіях, що використовуються. Це може зменшити різноманітність та непередбачуваність гонок, а також обмежити можливості гравця взаємодіяти з різними типами суперників.

Ураховуючи перелічені плюси та мінуси гонки Project CARS 2 з використанням ШІ у якості суперника, гравці можуть насолоджуватись реалістичною грою з інтелектуальними суперниками, але слід бути готовим до можливих технічних недоліків та обмежень, які можуть вплинути на загальний досвід гри.

1.4 Особливості гонки зі ШІ у якості суперника

Створення гонок зі штучним інтелектом (ШІ) у якості суперника відкриває нові можливості та особливості в гоночних іграх. ШІ виступає як контрольований комп'ютером опонент, який здатний адаптуватись до дій гравця та створювати реалістичні ситуації змагання.

Одна з ключових особливостей гонок з ШІ як суперником полягає в їхній інтелектуальній адаптивності. ШІ може аналізувати та враховувати дії гравця, його стратегію та здібності. На основі цих даних, ШІ може приймати рішення щодо своєї поведінки на трасі, намагаючись зробити її якомога більш складною та конкурентною. Це створює виклик для гравця, оскільки ШІ може адаптуватись до його стратегії та намагатись побити його.

Іншою важливою особливістю гонок з ШІ є їхня реалістичність. Розробники стараються створити ШІ, які діють подібно до реальних гонщиків. Вони моделюють різні аспекти гонки, такі як фізика автомобілів, поведінка на трасі та стратегії піт-стопу. Це дозволяє гравцям отримати максимально реалістичний досвід гоночних змагань та відчути себе насправді в гонках зі справжніми суперниками.

Також варто зазначити, що гонки з ШІ як суперником можуть бути налаштовані на різні рівні складності. Це дозволяє гравцям вибрати відповідний рівень виклику, починаючи від початківців і до досвідчених гравців. Такий гнучкий підхід дозволяє кожному гравцеві знайти свій комфортний рівень гри та насолоджуватись процесом змагань.

Однією з переваг гонок з ШІ як суперником є можливість грати одному гравцю без необхідності підключення до мережі або пошуку інших гравців. ШІ може виступати як суперник у будь-який час, що робить цю форму гри зручною та доступною для кожного.

Отже, розроблена гонка зі штучним інтелектом у якості суперника вносять особливий елемент у гоночні ігри, забезпечуючи гравцям інтелектуальний виклик та реалістичний досвід змагань. Завдяки адаптивності ШІ, їхній

реалістичності та можливості налаштування рівня складності, гонки з ШІ стають захоплюючими та цікавими для широкого кола гравців.

Враховуючи недоліки попередніх застосунків, розроблений у ході роботи застосунок може виділити деякі переваги, а саме низькі системні вимоги через використання інструментів Unity та просте текстурування, простий ігровий досвід, навіть недосвідчений гравець зможе з легкістю мати гарні результати у грі та не матиме проблем з опануванням керування машиною через примітивні налаштування керування машиною в грі.

ШІ інтелект має широкий спектр налаштувань які у майбутньому дозволять збільшити його складність на створити більш реалістичні та складні умови для гравця, що збільшить зацікавленість більш досвідчених гравців до застосунку.

1.5 Постановка завдання

Мета кваліфікаційної роботи полягає в створенні гонки з ШІ, яка буде використовувати алгоритми та штучний інтелект для пошуку оптимального шляху через невидимі маркери на трасі.

Основні вимоги до розробленої гонки з ШІ включають:

1. Реалістичність гоночної середовища: створення траси з невидимими маркерами, які ШІ повинен виявити та пройти через них у найкоротший час.
2. Алгоритми штучного інтелекту: розробка алгоритмів, які дозволять ШІ приймати рішення про найкращий шлях на основі зібраної інформації про маркери та поточне положення на трасі.
3. Графічний інтерфейс: розробка графічного представлення гонки з можливістю відображення траси, маркерів та положення ШІ.

Додаткові вимоги до проекту можуть включати можливість налаштування складності гонки, режими гри (одиночний гравець, змагання з іншими

ШІ або гравцями), систему підрахунку очок та досягнень, а також можливість збереження прогресу гри.

У процесі розробки гонки з ШІ, розробник повинен врахувати вищенаведені вимоги та застосувати відповідні алгоритми та технології для досягнення поставленої мети.

2 ДОСЛІДЖЕННЯ ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ ІГОР З ВИКОРИСТАННЯМ ШІ

2.1 Огляд технологій розробки ігор з використаннях ШІ

Одним із доступних рушіїв є Unreal Engine, розглянемо його переваги та недоліки у процесі розробки ігор з використанням ШІ:

Переваги:

Потужний движок: Unreal Engine надає широкий функціонал та високу продуктивність, що дозволяє створювати великі та складні ігрові світи зі зручним інтерфейсом для розробників.

Вбудована підтримка ШІ: Unreal Engine має вбудовану підтримку розробки ШІ, зокрема систему поведінки AI та готові компоненти для розумного поводження супротивників.

Велика спільнота: Unreal Engine має велику та активну спільноту розробників, що дозволяє знайти відповіді на запитання та отримати допомогу під час розробки ігор з ШІ.

Недоліки:

Складність: Unreal Engine є потужним інструментом, але його висока складність може вимагати більшої кількості часу та зусиль для освоєння та використання всіх можливостей.

Вимоги до апаратного забезпечення: Розробка на Unreal Engine може вимагати потужного комп'ютера для зручної роботи з графікою та обробкою складних об'єктів.

Також одним із популярних рушіїв для розробки ігор є GameMaker, після аналізу даних про цей рушій було виділено наступні переваги та недоліки:

Переваги:

Простота використання: GameMaker є легким у використанні інструментом, який дозволяє швидко створювати прості ігри з ШІ без необхідності глибоких знань програмування.

Гнучкість: GameMaker дозволяє розробникам вибирати різні підходи до реалізації ШІ, використовуючи скриптові мови або готові модулі та розширення.

Активна спільнота: GameMaker має велику спільноту розробників, що підтримується розробниками інструменту, що дозволяє отримати допомогу та поради під час розробки ігор з ШІ.

Недоліки:

Обмежені можливості: GameMaker може бути обмеженим у порівнянні з більш потужними движками, що можуть обмежувати розмаїтість та складність реалізації ШІ.

Обмежена підтримка: GameMaker має обмежену підтримку для розробки ШІ, що може призвести до обмеженості можливостей у створенні складних та розумних супротивників.

Unity є одним з найпопулярніших движків для розробки ігор, і він також надає широкі можливості для використання штучного інтелекту (ШІ) у якості сперника. Було детально розглянуто можливості, переваги та недоліки даного рушія.

Переваги Unity для розробки ігор з ШІ:

Зручне середовище розробки: Unity надає розробникам зручне та інтуїтивно зрозуміле середовище для розробки ігор з ШІ. Його інтерфейс є дружнім до користувача, що дозволяє швидко оволодіти основами та прискорити процес розробки.

Вбудована підтримка ШІ: Unity має вбудовані інструменти та бібліотеки для розробки ШІ. Він надає можливість створювати складну поведінку супротивників, використовуючи різноманітні алгоритми та техніки ШІ, такі як машинне навчання, розподілені системи та багато іншого.

Мультиплатформеність: Unity підтримує розробку ігор для різних платформ, таких як ПК, консолі, мобільні пристрої та віртуальна реальність. Це дозволяє розробникам створювати ігри з ШІ, які доступні для широкої аудиторії користувачів.

Активна спільнота: Unity має велику та активну спільноту розробників, що сприяє обміну знаннями, досвідом та розробкою спільних рішень. Це дозволяє розробникам швидко знаходити відповіді на запитання, розв'язувати проблеми та отримувати підтримку під час розробки ігор з ШІ.

Інтегроване середовище розробки: Unity надає зручне та інтуїтивно зрозуміле середовище розробки, яке дозволяє розробникам швидко створювати скрипти, візуалізувати об'єкти, керувати анімацією та іншими аспектами гри. Це полегшує розробку ШІ, оскільки можна швидко прототипувати та тестувати поведінку супротивників.

Широке застосування: Unity широко використовується в індустрії розробки ігор та має вражаючий рекорд успіху. Велика кількість популярних ігор, включаючи ті, де використовується ШІ, була створена на Unity. Це свідчить про його потужність, надійність та ефективність.

Недоліки Unity для розробки ігор з ШІ:

Обмежені можливості ШІ: Хоча Unity надає вбудовані інструменти для розробки ШІ, вони можуть бути обмеженими у порівнянні зі спеціалізованими бібліотеками та фреймворками ШІ. Розробники можуть стикатися з обмеженостями при реалізації складних алгоритмів та поведінки супротивників.

Вимоги до продуктивності: Розробка складних ігор з ШІ в Unity може вимагати значних ресурсів обчислювальної потужності та пам'яті. Для досягнення високої продуктивності та оптимальної роботи ШІ, може бути необхідно провести оптимізацію та тестування гри.

Вартість ліцензії: Unity має безкоштовну версію, але деякі функціональні можливості та додаткові інструменти доступні тільки за плату. Це може вплинути на вартість розробки ігор з ШІ та бюджет проекту.

У висновку використання Unity для розробки ігор з ШІ має безліч переваг, включаючи багатоплатформеність, зручне середовище розробки, широкий функціонал, підтримку спільноти та широке застосування в індустрії. Ці переваги роблять Unity вигідним вибором для розробників, які прагнуть створити ігри з захоплюючим ШІ та досягти успіху на ринку геймдеву.

2.2 Види ШІ застовуємого в різних іграх у виді супротивника

Штучний інтелект (ШІ) в іграх використовується для створення супротивників, які здатні досконало взаємодіяти з гравцем та надати цікавий та викликаючий геймплей-досвід. Кожен тип ШІ супротивника має свої відмінності та особливості, які впливають на їхню поведінку та рівень інтелектуальності. Давайте розглянемо докладніше кожен вид ШІ супротивників у іграх.

Реактивний ШІ (Reactive AI)

Реактивний ШІ — це тип ШІ, який використовує прості алгоритми реагування на дії гравця. Супротивники з реактивним ШІ можуть розпізнавати конкретні дії гравця і виконувати певні дії відповідно. Наприклад, якщо гравець атакує супротивника, він може виконати блокування або ухиляння від удару. Основна особливість реактивного ШІ - це його простота. Він швидко реагує на конкретні ситуації, але може бути менш адаптивним до змін у геймплеї та менш здатним до стратегічного мислення.

Плануючий ШІ (Planned AI)

Плануючий ШІ — це тип ШІ, який використовує складні алгоритми планування та прийняття рішень. Супротивники з плануючим ШІ можуть аналізувати ситуацію, прогнозувати можливі рухи гравця та створювати стратегію для досягнення своїх цілей. Вони можуть розробляти довгострокові плани та вибирати найкращі дії для досягнення успіху в грі. Плануючий ШІ може бути більш складним та інтелектуальним у порівнянні з реактивним ШІ, але він також може бути витратним за ресурсами та часом, оскільки вимагає багато обчислень для прийняття оптимальних рішень.

Еволюційний ШІ (Evolutionary AI)

Еволюційний ШІ — це тип ШІ, який використовує принципи еволюції для покращення своїх навичок. Він базується на генетичних алгоритмах, які здатні відбирати найкращі стратегії та комбінації дій з популяції супротивників. Еволюційний ШІ може самостійно навчатися та вдосконалюватися з кожною грою, адаптуватися до стилю гравця та вибудовувати непередбачувані

стратегії. Він може бути дуже гнучким та здатним до адаптації до змін у геймплеї та стилі гравця. Однак, цей тип ШІ може вимагати багато часу для навчання та прогресування.

Ці три основні типи ШІ супротивників представляють різні підходи до створення інтелектуального поведінки. Вибір конкретного типу ШІ залежить від цілей розробників гри, їх ресурсів та сподіваного рівня складності для гравців. Кожен тип ШІ має свої переваги та обмеження, і використання правильного підходу може забезпечити захоплюючий та викликаючий геймплей для гравців. З розвитком технологій ШІ ми можемо очікувати ще більш складних та реалістичних супротивників у майбутніх іграх.

2.3 Огляд методів імплементування ШІ як суперника в ігри на Unity

Розробка ігор є складним процесом, який вимагає багато уваги до деталей, зокрема до розробки інтелектуального супротивника (ШІ). ШІ відповідає за поведінку та прийняття рішень супротивника в грі, що забезпечує цікавість та виклик для гравця. У середовищі Unity, одного з найпопулярніших двигунів для розробки ігор, існує кілька методів імплементації ШІ як суперника. Розглянемо деякі з них та їх відмінності та особливості.

Скінченний автомати (Finite State Machines)

Скінченний автомат — це модель поведінки, яка складається зі скінченної кількості станів, переходів між станами та дій, пов'язаних з кожним станом. Використання скінченних автоматів для імплементації ШІ дозволяє задати різні стани, наприклад, "атака", "відступ", "патрулювання" тощо, та визначити логіку переходу між станами. Кожен стан може мати свої власні правила та дії, що робить поведінку суперника контрольованою та передбачуваною. Скінченні автомати дозволяють реалізувати прості та середньо складні стратегії супротивника, але можуть бути обмеженими у вирішенні складних ситуацій та адаптації до змін у грі.

Нейронні мережі (Neural Networks)

Нейронні мережі — це комп'ютерна модель, яка намагається імітувати роботу людського мозку. Використання нейронних мереж для імплементації ШІ дозволяє створювати більш складну та гнучку поведінку суперника. Нейронна мережа може навчатися на основі даних з гри та покращувати свої рішення з часом. Вона може адаптуватися до нових ситуацій, реагувати на зміни у грі та навіть виробляти стратегії, які не були задані розробником. Однак, розробка та навчання нейронних мереж можуть бути часо- та ресурсомісткими процесами.

Алгоритми пошуку (Search Algorithms)

Алгоритми пошуку, такі як алгоритм Мінімакс або алгоритм A*, використовуються для прийняття рішень суперника на основі оцінки стану гри та визначення найкращого кроку. Ці алгоритми аналізують можливі ходи та їх наслідки, шукаючи найоптимальнішу стратегію. Вони забезпечують супернику здатність приймати обґрунтовані рішення, розраховуючи на кращий можливий результат. Використання алгоритмів пошуку може бути ефективним, але вимагає обчислювальних ресурсів та може бути складним у використанні для деяких типів ігор.

Поведінкові дерева (Behavior Trees)

Поведінкові дерева є ієрархічною структурою, яка описує поведінку суперника. Вони складаються з вузлів, що представляють різні дії або рішення, та визначають логіку переходу між ними. Поведінкові дерева дозволяють організовувати складну поведінку суперника, розгалужувати логіку на основі умов та задавати пріоритети для різних дій. Використання поведінкових дерев може бути зручним для моделювання складних стратегій супротивника, але вимагає великої кількості вузлів та правил.

Вибір методу імплементації ШІ як суперника в іграх на Unity залежить від типу гри, вимог до поведінки суперника та обмежень ресурсів. Кожен метод має свої відмінності та особливості, і розробник повинен обирати той, який найкраще відповідає потребам конкретної гри.

Необхідно також враховувати, що комбінування різних методів імплементації ШІ може призвести до досягнення більшої реалістичності та складності поведінки суперника. Застосування методів машинного навчання разом з традиційними алгоритмами може дати кращі результати в розробці інтелектуального супротивника.

Усі ці методи імплементації ШІ в Unity є потужними інструментами для створення різноманітних супротивників в іграх. Розробники повинні аналізувати вимоги гри та вибрати найкращий підхід для досягнення потрібного рівня інтелектуальності та поведінки суперників.

2.4 Засоби розробки ігор на Unity

Сцена (Scene)

Сцена є основною одиницею розробки в Unity. Вона визначає простір, в якому відбувається гра, і містить різні об'єкти, такі як персонажі, об'єкти оточення, світло, звук і т.д. У сцені можна створювати, редагувати та організовувати об'єкти за допомогою візуального редактора Unity. Кожна сцена може мати свої налаштування, скрипти та ресурси.

Об'єкт (GameObject)

Об'єкт є основною будівельною одиницею гри в Unity. Він може представляти персонажа, предмет, транспортний засіб або будь-який інший елемент гри. Об'єкти можуть мати компоненти, які визначають їх поведінку, вигляд та взаємодію з іншими об'єктами. Наприклад, компонент Rigidbody визначає фізичну поведінку об'єкта, а компонент Mesh Renderer відповідає за візуальне відображення об'єкта на екрані.

Скрипт (Script)

Скрипти в Unity використовуються для програмування поведінки об'єктів та керування логікою гри. Unity підтримує мови програмування, такі як C# та JavaScript, для написання скриптів. Скрипти можна прикріплювати до об'єктів у сцені, і вони виконуватимуться під час гри, взаємодіючи з об'єктами та виконуючи потрібні операції.

Фізика (Physics)

Unity має вбудовану систему фізики, яка дозволяє моделювати реалістичну фізичну поведінку об'єктів у грі. Розробники можуть встановлювати фізичні властивості об'єктів, такі як маса, сила тяжіння, колізії та інерція. Це дозволяє створювати реалістичні ефекти руху, зіткнень та інтеракцій між об'єктами.

Анімація (Animation)

Unity надає інструменти для створення анімацій об'єктів у грі. За допомогою анімаційних систем Unity, розробники можуть створювати рухи, трансформації та інтерактивні ефекти для персонажів та об'єктів. Це дозволяє живо відтворювати рухи персонажів, створювати ефекти зміни стану та інші динамічні елементи гри.

Матеріали та текстури (Materials & Textures)

Unity підтримує широкий спектр матеріалів та текстур, які використовуються для візуального оформлення об'єктів у грі. Розробники можуть створювати текстури для об'єктів, наносити їх на поверхні об'єктів, а також налаштовувати властивості матеріалів, такі як колір, блік та прозорість.

Unity Asset Store

Це онлайн-магазин, де розробники можуть придбати готові ресурси, такі як графічні моделі, звукові ефекти, скрипти, плагіни та інше. Asset Store допомагає економити час та зусилля, надаючи доступ до великого вибору готових ресурсів, які можна використовувати у своїх проектах.

Unity Collaborate

Це інструмент для спільної роботи над проектами у команді. Він надає можливість розробникам обмінюватися змінами, керувати версіями проекту, вирішувати конфлікти інтеграції та співпрацювати над одним проектом в реальному часі.

Unity Cloud Build

Це хмарний сервіс, який автоматизує процес збірки, тестування та розгортання проектів Unity на різних платформах. Розробники можуть налаштовувати конфігурації збірки, і сервіс автоматично буде здійснювати збірку проекту за змінами, що дозволяє ефективно керувати процесом розробки та тестування.

Unity Analytics

Це інструмент, що дозволяє розробникам збирати, аналізувати та отримувати відомості про поведінку гравців у грі. За допомогою Unity Analytics, розробники можуть здійснювати вимірювання успішності гри, вивчати взаємодію гравців з ігровими елементами та отримувати аналітичні дані для вдосконалення геймплею та монетизації.

Unity Multiplayer

Це фреймворк для створення мультиплеєрних ігор. Він надає розробникам інструменти для створення мережевої взаємодії між гравцями, синхронізації даних та керування мультиплеєрними сеансами. За допомогою Unity

Multiplayer, можна створювати як локальні мережеві ігри, так і ігри з підтримкою глобального мультиплеєра через Інтернет.

Visual Studio ma Visual Studio Code

Ці інтегровані середовища розробки (IDE) дозволяють програмістам писати код для проектів Unity з використанням різних мов програмування, таких як C# або JavaScript. Вони надають розширені можливості редактора коду, дебагери та інструменти для підсвічування синтаксису та автодоповнення, що полегшує розробку програмного забезпечення.

Blender ma Autodesk Maya

Ці програми для 3D-моделювання дозволяють розробникам створювати графічні об'єкти, персонажів, ландшафти та анімації для використання у проектах Unity. Вони мають різноманітні інструменти для моделювання, текстурування, риггінгу та анімації, що дозволяє створювати вражаючі візуальні ефекти та живі світи у ваших іграх.

ProBuilder

Це інструмент для швидкого створення та редагування 3D-моделей прямо всередині Unity. ProBuilder надає можливість швидко будувати примітиви, модифікувати їх форму, додавати деталі та оптимізувати геометрію моделей без потреби використовувати зовнішні програми для моделювання.

3 РОЗРОБКА ГОНОЧНОЇ ГРИ З ШІ У ЯКОСТІ СУПЕРНИКА

3.1 Призначення розробки

Призначення розробки ігрового гоночного застосунку на Unity полягає у створенні високоякісного віртуального середовища, в якому гравці можуть насолоджуватись захоплюючими гонками та отримувати задоволення від ігрового процесу. Визначити переваги використання Unity як рушія для створення ігор та імплементування ШІ як суперника. Виконати тестування застосунку та можливостей ШІ у різних ігрових умовах.

3.2 Функціональні та нефункціональні вимоги

Функціональні вимоги до застосунку:

Розроблений в ході виконання дослідження ігровий застосунок надавати наступні функціональні вимоги:

1. Користувач повинен мати змогу запустити ігровий застосунок.
2. Користувач повинен мати змогу закрити ігровий застосунок.
3. Користувач може обирати різні пункти у головному меню.
4. Користувач може обирати різні кольори машин, режими гри та різні траси.
5. Користувач має можливість провести гонку з ШІ у якості суперника.

Діаграма use-case по використанню ігрового застосунку користувачем зображена на рис.4.

Нефункціональні вимоги до застосунку:

1. Зручний та зрозумілий інтерфейс ігрового застосунку.
2. Уся взаємодія з застосунком відбувається через внутрішньоігровий інтерфейс.
3. Простота у використанні.

4. Низькі системні вимоги до користувача.
5. Забезпечення користувача можливістю зручно навігувати по меню застосунку, вільно обирати режими та отримувати ігровий досвід.

3.3 Проектування застосунку

Після розглядання засобів та можливостей рушія Unity, було спроектовано прототип ігрового застосунку.

Серед доступних функцій застосунок надає можливість обирати різні режими гри, траси та кольори машини гравця. Наступна діаграма (Рис.4) демонструє можливості програми, розробленої у рушії Unity з написанням скриптів у Visual Studio 2017, і передбачає використання спроектованого в Unity інтерфейсу в якості головного меню, для забезпечення користувача можливістю орієнтуватися та обирати необхідні функції та налаштування застосунку.

Діаграма демонструє можливість застосунку попередньо надати вибір користувачу, який колір машини, режим гри та трасу він обере.

Після вибору усіх цікавлячих гравця пунктів він перейде до самого ігрового процесу. По завершенню гонки у гравця буде можливість вийти з ігрового застосунку.

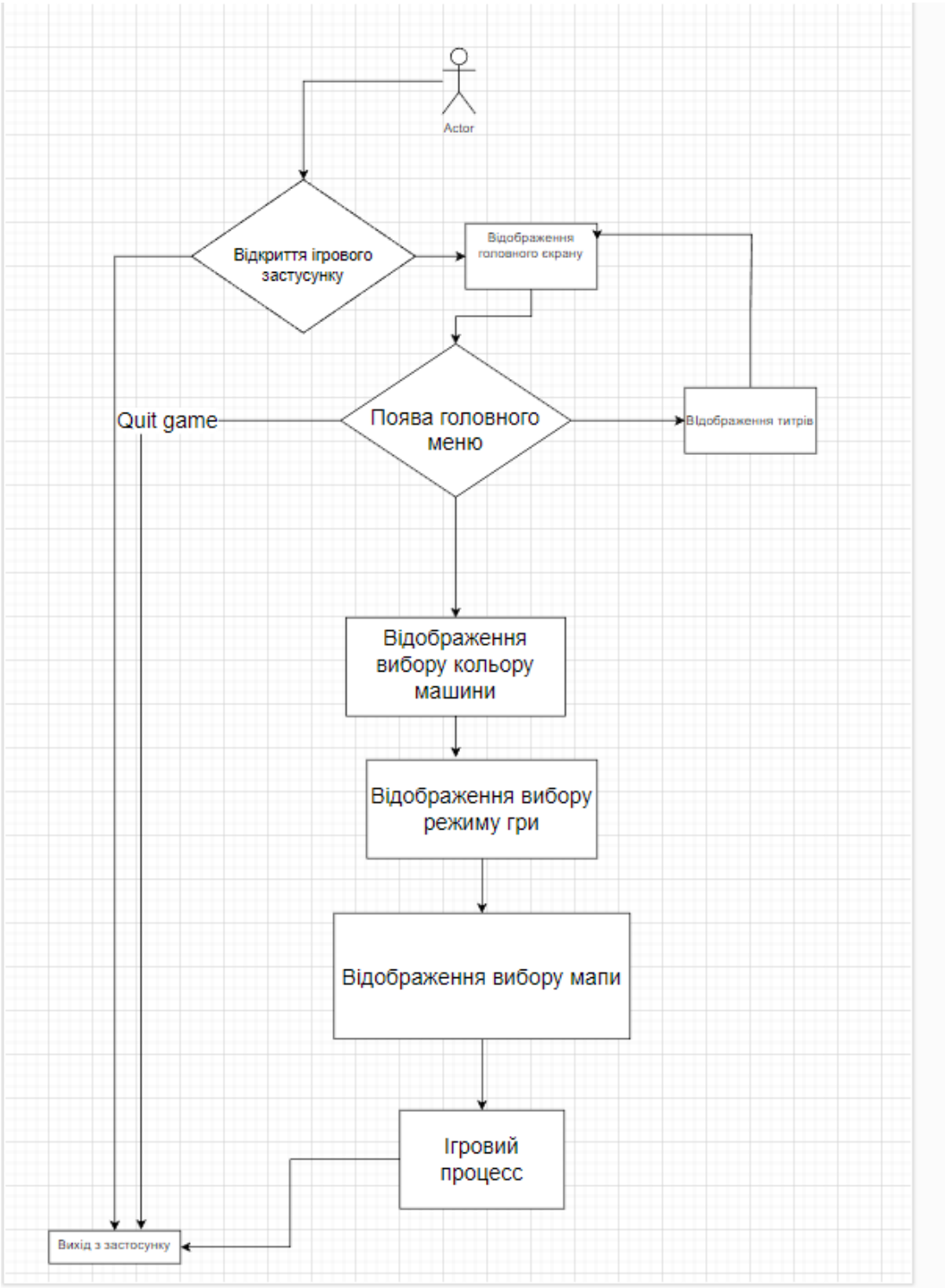

Рис.*4 Use-case діаграма*

Був спроектований ШІ у якості суперника у ігровому застосунку заради надання гравцю більш реалістичного ігрового досвіду.

ШІ буде активуватись після старту гри, та починати пошук невидимих маркерів розташованих вздовж траси (Рис. 18).

При колізії з гравцем або перешкодою ШІ буде обробляти вхідні дані про положення тих чи інших об`єктів відносно нього та приймати рішення щодо подальших дій — піти на маневр або зменшити швидкість руху щоб запобігти зіткненню.

Після урахування усіх факторів та реакції на них ШІ знову продовжуватиме пошук маркерів для продовження гонки та отримання перемоги (Рис.5).

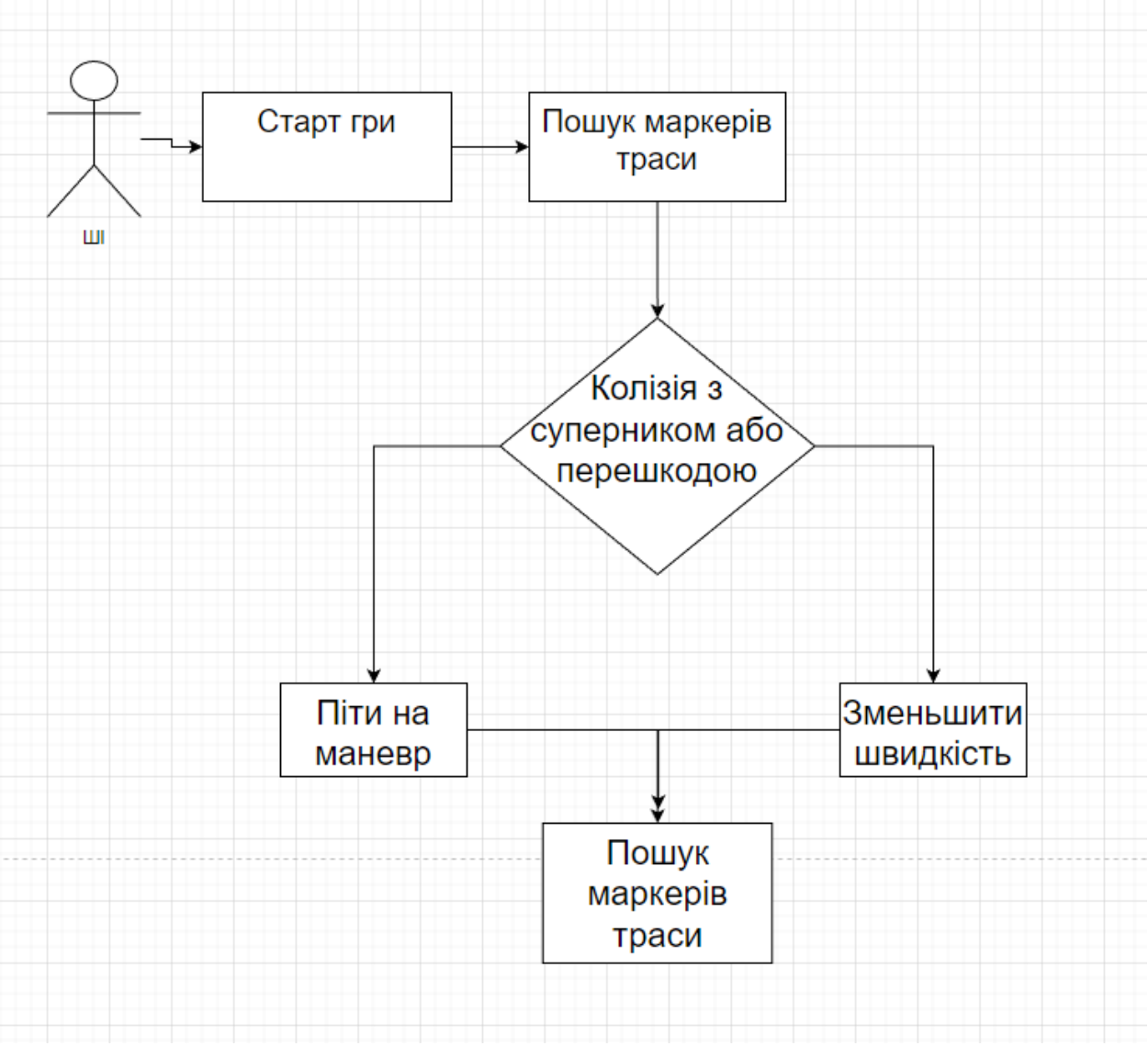

Рис.5 *Діаграма станів ШІ*

3.4 Засоби реалізації

Побудований ігровий застосунок був створений у ігровому рушії Unity з використанням його вбудованих можливостей для моделювання трас да дизайну меню. Скрипти для функціоналу меню, ігрових режимів та поведінки ШІ були створені у середовищі розробки Visual Studio 2017, об’єктно орієнтованою мовою C#.

Unity — це потужне середовище розробки для створення ігрових застосунків. У розробці гоночного застосунку з використанням ШІ у якості суперника ви можете використовувати різні класи та методи, щоб реалізувати функціональність гри. Ось деякі з них:

Клас "CarController" (Контролер автомобіля):

Методи для керування автомобілем, такі як "Steer" (рух вбік), "Accelerate" (прискорення), "Brake" (гальмування) і "Reset" (перезапуск).

Методи для перевірки стану автомобіля, наприклад, "IsMoving" (перевірка чи автомобіль рухається) і "IsCrashed" (перевірка чи автомобіль зіткнувся).

Клас "AIController" (Контролер ШІ):

Методи для реалізації прийняття рішень ШІ, такі як "MakeDecision" (прийняття рішення про наступний крок), "AvoidObstacles" (уникнення перешкод) і "ChooseTarget" (вибір цілі).

Методи для контролю руху ШІ, наприклад, "FollowPath" (переслідування маршруту) і "AdjustSpeed" (налаштування швидкості).

Клас "RaceManager" (Менеджер гонки):

Методи для керування гоночними подіями, такі як "StartRace" (початок гонки), "FinishRace" (закінчення гонки) і "CalculateResults" (обчислення результатів).

Методи для сповіщення гравців і ШІ про події гонки, наприклад, "CheckpointReached" (сповіщення про досягнення контрольної точки) і "NotifyCollision" (сповіщення про зіткнення).

Клас "TrackSelect" (Вибір мапи):

Методи для визначення геометрії траси, наприклад, "CreateTrackMesh" (створення мешу траси) і "CalculateCurve" (обчислення кривої траси).

Методи для розміщення об'єктів на трасі, такі як "SpawnObstacles" (розміщення перешкод) і "ColourScore" (видача очок).

Unity також надає можливість розробляти скрипти на мові програмування C#. Ця мова є потужним інструментом для створення функціональності в ігрових застосунках. Ось деякі класи та методи, які можна використовувати при розробці скриптів на C# в Unity:

Клас "MonoBehaviour":

Метод "Start" — викликається один раз при запуску скрипту.

Метод "Update" — викликається кожен кадр і використовується для оновлення стану об'єкта.

Клас "Transform":

Методи "Translate" — зміщує об'єкт у просторі за певним вектором.

Методи "Rotate" — поворот об'єкта на певний кут.

Клас "Input":

Методи "GetAxis" — отримання значення вісі управління (наприклад, газ, гальма, кермо).

Методи "GetButton" — перевірка натискання кнопки на клавіатурі або контролері.

Клас "Collision":

Метод "OnCollisionEnter" — викликається, коли об'єкт зіштовхнеться з іншим об'єктом.

Метод "OnCollisionExit" — викликається, коли об'єкт вийде зі стану зіткнення з іншим об'єктом.

Класи для роботи з графікою та анімацією:

Клас "SpriteRenderer" - використовується для відображення спрайтів.

Клас "Animator" - використовується для управління анімацією об'єктів.

Під час розробки ігрового застосунку на Unity та написанні скриптів на C# також використовуються різноманітні бібліотеки для розширення функціоналу та спрощення процессу розробки застосунку:

Unity Standard Assets: Ця бібліотека надає готові компоненти та ресурси, такі як моделі персонажів, ефекти світла, звуки, контролери руху та багато іншого. Вона може значно спростити створення різноманітних елементів гри.

Unity UI: Ця бібліотека дозволяє швидко створювати інтерфейс користувача (UI) для вашої гри. Вона містить різноманітні компоненти, такі як кнопки, тексти, полоски прогресу та інші, для створення зручного та привабливого інтерфейсу.

Cinemachine: Ця бібліотека надає потужні інструменти для керування камерою в грі. Вона дозволяє створювати складні камерні рухи, змінювати положення та орієнтацію камери залежно від дії гравця або подій в грі.

Unity Engine API: Це основна бібліотека Unity, яка надає доступ до всіх основних функцій та можливостей движка. Вона містить класи та методи для роботи з графікою, анімацією, фізикою, звуком, управлінням та багатьма іншими аспектами розробки ігор.

UnityEngine.UI: Ця бібліотека містить класи та методи для роботи з інтерфейсом користувача. Вона дозволяє створювати кнопки, тексти, зображення та інші елементи інтерфейсу, а також взаємодіяти з ними.

UnityEngine.Networking: Ця бібліотека дозволяє створювати мультиплеєрні ігрові застосунки. Вона надає класи та методи для створення з'єднання між гравцями, передачі даних та синхронізації стану гри.

UnityEngine.AI: Ця бібліотека надає функціональність для роботи зі штучним інтелектом (ШІ) в Unity. Вона містить класи та методи для реалізації розумного поведінки супротивників у грі, включаючи навігацію, пошук шляху та вирішення колізій.

System.Collections: Ця бібліотека включає різні класи та інтерфейси для роботи з колекціями даних, такими як списки, словники, стеки тощо. Вона

надає зручний спосіб управління та маніпулювання даними в ігровому застосунку.

3.5 Розробка застосунку

3.5.1 Системні вимоги до розробленої гри

- Операційна система: Windows 7 або новіша, або Mac OS X 10.9 або новіша.
- Процесор: Двоядерний процесор з тактовою частотою 2.0 ГГц або еквівалентний.
- Оперативна пам'ять: 4 ГБ.
- Графічна карта: З підтримкою DirectX 11 або OpenGL 3.3 із 1 ГБ відеопам'яті.
- Вільне місце на жорсткому диску: 5 ГБ.
- Звукова карта: Сумісна з DirectX.
- Інші: Клавіатура та миша, доступ до Інтернету для онлайн-ігор або оновлень.

3.5.2 Встановлення усіх необхідних компонентів

Перший крок — це встановлення Unity Hub та усіх необхідних бібліотек для розробки застосунку. На приведеному нижче рисунку 12 зображені усі компоненти, які необхідно встановити для повного функціонування та доступу до усіх необхідних інструментів, які будуть використані у процесі розробки, а саме рушій версії 5.4.2f2, середовище розробки Visual Studio 2017 та підтримка різних платформ (див. Рис.6).

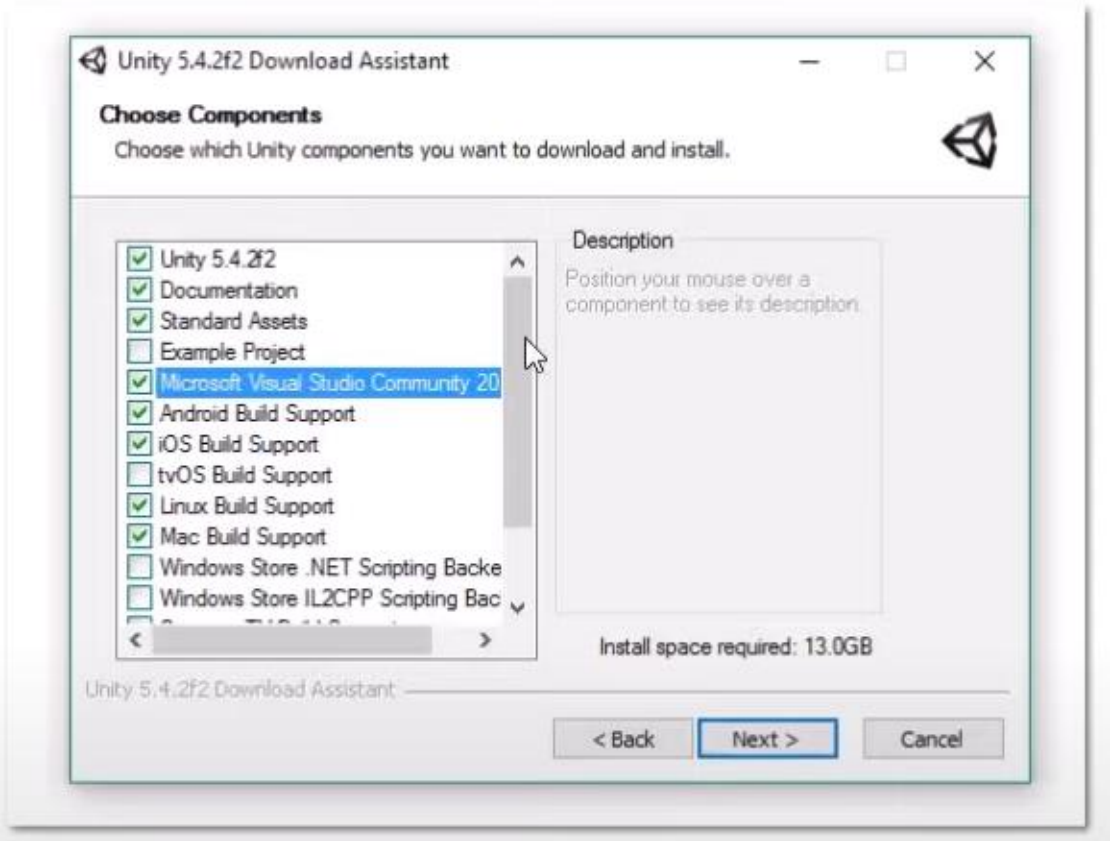

Рис.*6 Компоненти Unity*

3.5.3 Керівництво для користувача

При відкритті .exe файлу гри користувач бачить інтерактивне вікно з можливістю вибору налаштувань графіки, розміру екрану, вибору екранного режиму, дисплею та інструкцію щодо керування (див. Рис.7) .

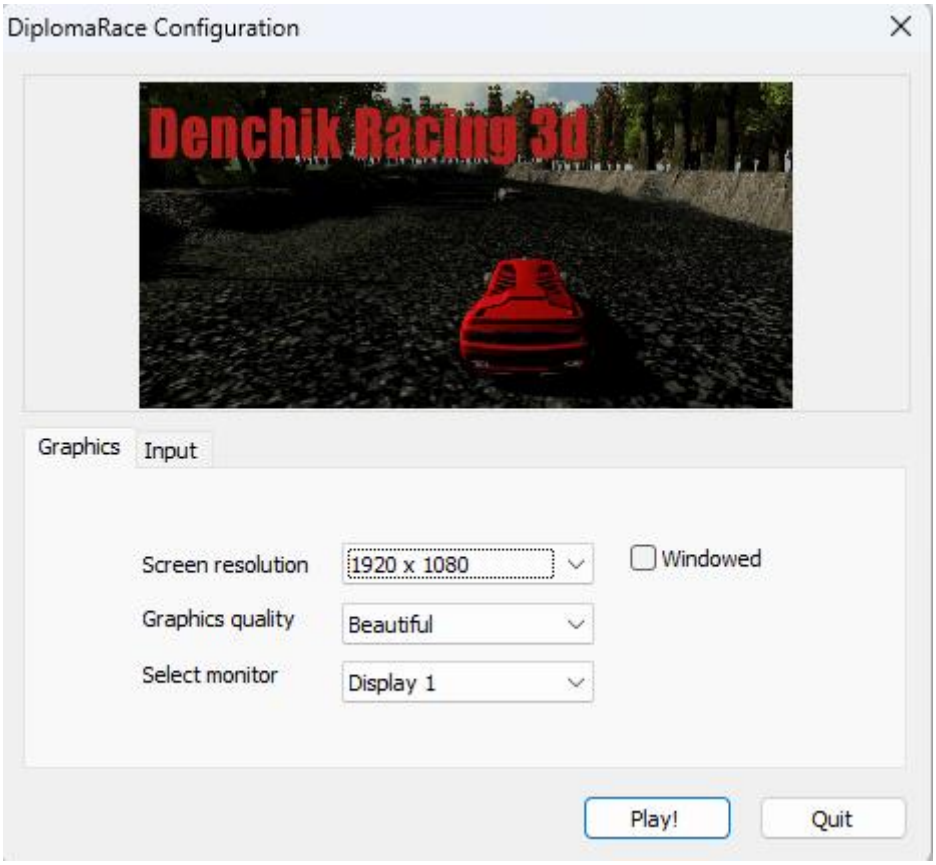

Рис.7 Меню первинних налаштувань

Розробник сам має можливість налаштувати зовнішній вигляд ярлику на лого гри у меню. Лого було створено на основі зображень з гри та при допомозі графічного редактору Photoshop, після чого було додано в список текстур та розміщено в меню (див.Рис.8).

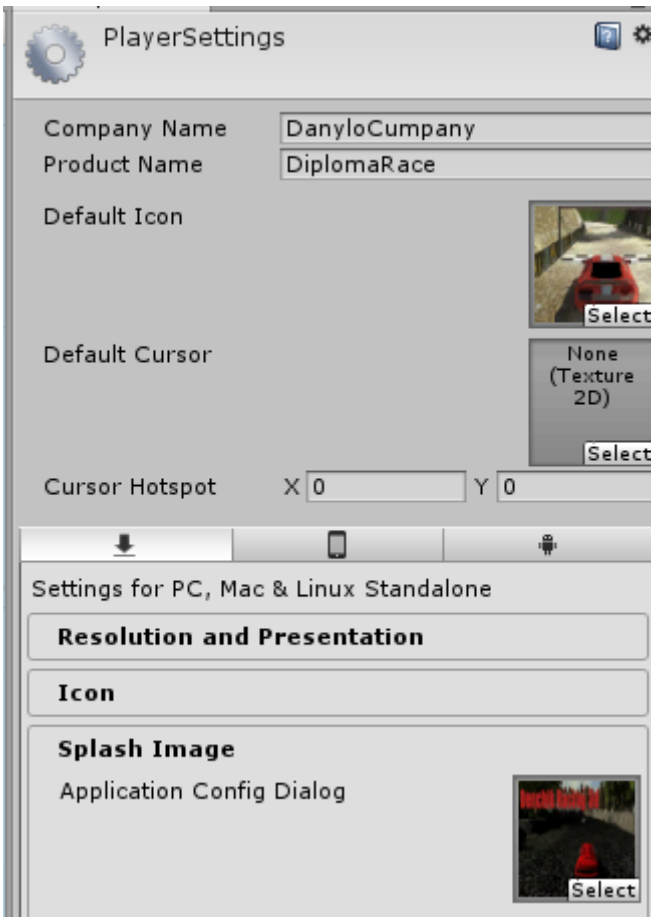

Рис.*8 Вибір текстур в меню*

Після вибору усіх налаштувань користувач натискає Play, відкривається сам застосунок та користувач бачить головний екран (див.Рис.9) (Див. Лістинг 1, 2). Користувачу надається підказка натиснути будь-яку кнопку для переходу в меню гри з усім функціоналом (див. Рис.10) (Див. Лістинг 3).

Зміна стану GameObject largeButton з false на true при реакції на натискання користувачем переводить користувача у головне меню за допомогою SceneManager (Лістинг 3). Рух фону головного екрану реалізований за допомогою надання елементам фону параметру rotateSpeed (Лістинг 2).

Лістинг *1 головний екран*

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

```
public class MenuAppear : MonoBehaviour {

    public GameObject largeButton;
    public GameObject textClick;
    public GameObject menuButtons;

    public void StartMenu()
    {
        textClick.SetActive(false);
        menuButtons.SetActive(true);
        largeButton.SetActive(false);
    }
```

Лістинг *2 рух фону головного екрану*

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

public class SkyRot : MonoBehaviour {

    public float rotateSpeed = 0.5f;

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

Лістинг *3 відображення наступного екрану меню*

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

public class ButtonOption : MonoBehaviour {
     public void PlayGame () {
```

```
        SceneManager.LoadScene (2);
    }
    public void MainMenu () {
        SceneManager.LoadScene (1);
    }
    public void Track01 () {
        SceneManager.LoadScene (3);
    }
    public void Track02 () {
        SceneManager.LoadScene (4);
    }
   public void CreditScene()
   {
       SceneManager.LoadScene(5);
   }
}
```

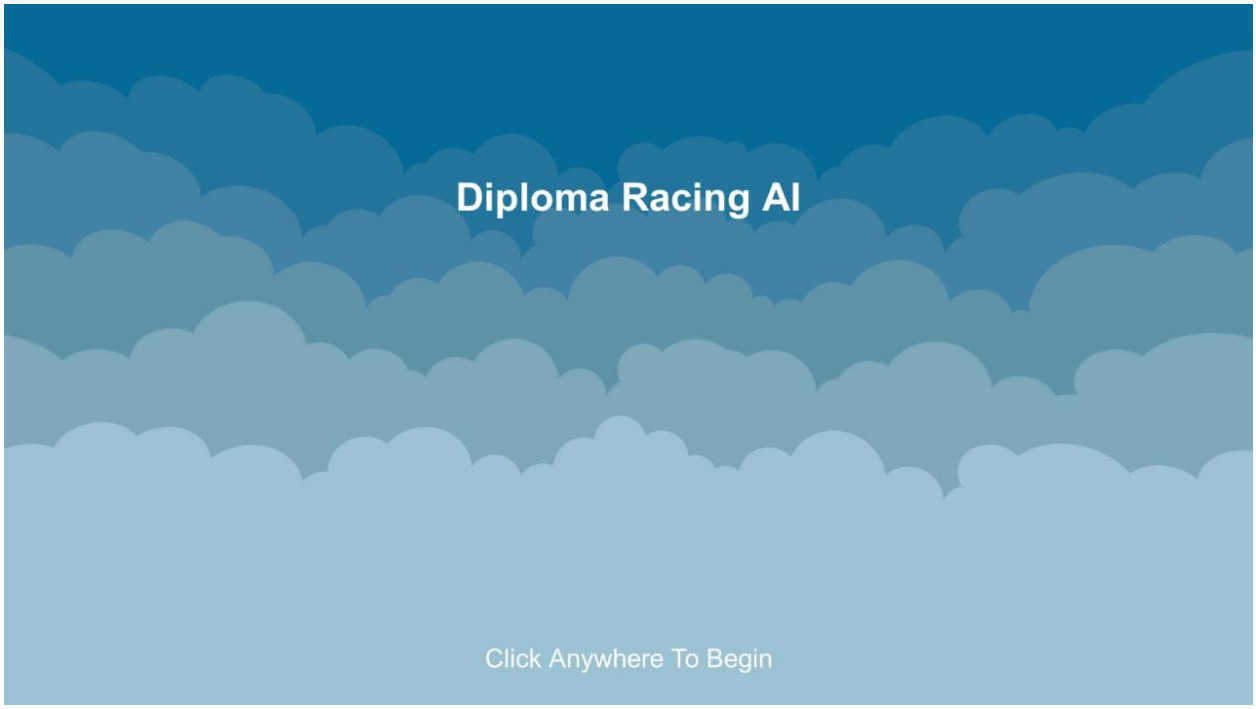

Рис.9 *головний екран*

Рис.*10 Головне меню*

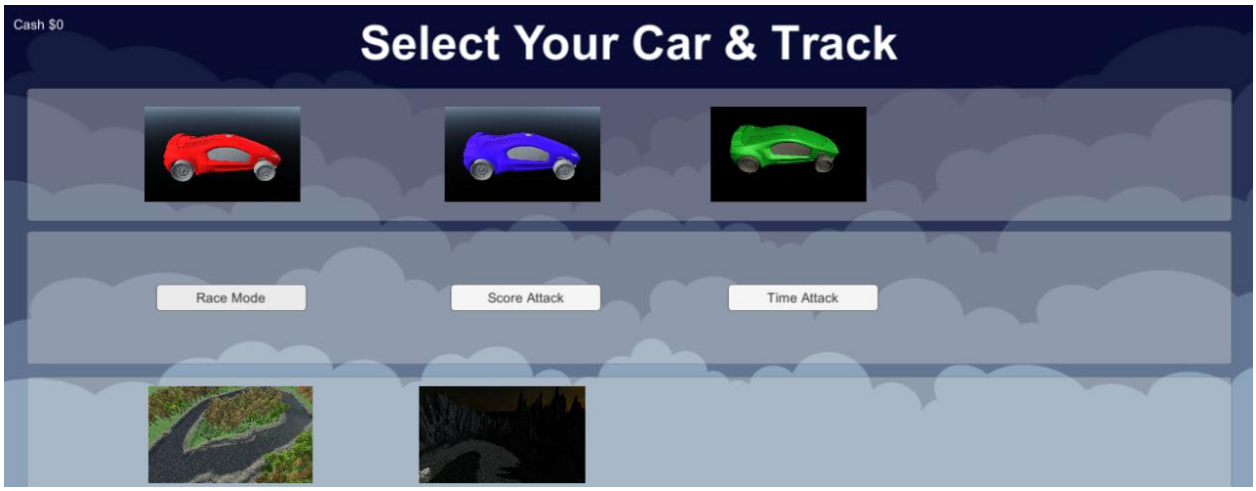

Рис.*11 Вибір налаштувань ігрового режиму*

Кнопка Play Game також реагує на натискання, після зміни стану на true запускається SceneManager та переводить гравця у меню вибору налаштувань ігрового режиму.

Налаштування відкриваються одне за іншим після вибору попереднього (Див.Рис.9) (Див. Лістинг 4).

Лістинг *4 Вибір кольору автомобіля*

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

```
using UnityEngine;

public class CarChoice : MonoBehaviour {

    //1=Red, 2=Blue
    public GameObject RedBody;
    public GameObject BlueBody;
    public GameObject GreenBody;
    public int CarImport;

     void Start () {
        CarImport = GlobalCar.CarType;
        if (CarImport == 1)
        {
            RedBody.SetActive(true);
        }

        if (CarImport == 2)
        {
            BlueBody.SetActive(true);
        }

        if (CarImport == 3)
        {
            GreenBody.SetActive(true);
        }
    }

}
```

Наступний крок — це вибір режиму гри (Див. Лістинг 5), їх представлено 3, це Race Mode, Score Attack та Time Attack.

Race Mode – Звичайний режим гри в якому гравцю потрібно пройти 3 кола трасси та фінішувати першим в останньому колі для виграшу над суперником. Режим починається з відліку в 3 секунди (Див. Лістинг 9), після того як гравець або ШІ проходить половину трасси активується тріггер фінішу кола (Див. Лістинг 6), після фінішу кола тріггер знову деактивується до того як один з гравців не пройде половину кола знову, після пройдення трьох кіл перший зробивший це гравець визнається переможцем, запускається анімація перемоги та відповідна музика (Див. Лістинг 10).

Score Attack – У цьому режимі гравець отримує очки за збирання деталей різних кольорів (Дивю Лістинг 7), у залежності від кольору гравець отримує різну кількість ігрових очок, по завершенню гонки гравець отримеє результат у вигляді очок які відображаються у панелі Score (див.рис.12).

Time Attack – Режим на одного гравця суть якого полягає у найшвидшому проходженні траси, таймер (Див. Лістинг 8) розташований на екрані та відображає час який гравець витрачає на проходження гонки, найкращий час відображається на екрані під самим таймером (див.рис.13).

3.5.4 Програмна реалізація застосунку

В лістингу 5 відображено клас вибору режиму гри, при активації 1 з режимів він завжди змінює TrackSelect.SetActive на true, що відкриває гравцю доступ до вибору мапи.

Лістинг 5 *Вибір режиму гри*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class ModeSelect      : MonoBehaviour {
    public static int RaceMode; // 0=Race, 1=Score, 2=Time
    public GameObject TrackSelect;
    public void ScoreMode ()
    {
```

```
        RaceMode = 1;
        TrackSelect.SetActive(true);
    }
    public void TimeMode()
    {
        RaceMode = 2;
        TrackSelect.SetActive(true);
    }
    public void TheRaceMode()
    {
        RaceMode = 0;
        TrackSelect.SetActive(true);
    }
}
```

У лістингу 6 відображено скрипт відповідаючий за активацію тріггеру пройдення половини кола та активацію фінішної лінії. При колізії з HalfLapTrig активується LapCompleteTrig відповідаючий, при проходженні якого змінюється кількість пройдених кіл та зберігається час який було витрачено на проходження кола у LapTimeManager який наділений параметрами збереження часу до мілісекунд. Сам таймер та запис часу у режимі гри реалізований у Лістингу 8.

Лістинг *6 тріггери пройдення половини кола та фінішу*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class HalfPointTrigger : MonoBehaviour {
    public GameObject LapCompleteTrig;
    public GameObject HalfLapTrig;
    void OnTriggerEnter () {
        LapCompleteTrig.SetActive (true);
        HalfLapTrig.SetActive (false);
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class LapComplete : MonoBehaviour {
     public GameObject LapCompleteTrig;
     public GameObject HalfLapTrig;
     public GameObject MinuteDisplay;
     public GameObject SecondDisplay;
     public GameObject MilliDisplay;
     public GameObject LapTimeBox;
     public GameObject LapCounter;
     public int LapsDone;
     public float RawTime;
     public GameObject RaceFinish;
     void Update () {
          if (LapsDone == 2) {
               RaceFinish.SetActive (true);
          }
     }
     void OnTriggerEnter () {
          LapsDone += 1;
          RawTime = PlayerPrefs.GetFloat ("RawTime");
          if (LapTimeManager.RawTime <= RawTime) {
               if (LapTimeManager.SecondCount <= 9) {
                    SecondDisplay.GetComponent<Text> ().text
= "0" + LapTimeManager.SecondCount + ".";
               } else {
                    SecondDisplay.GetComponent<Text> ().text
= "" + LapTimeManager.SecondCount + ".";
               }

               if (LapTimeManager.MinuteCount <= 9) {
                    MinuteDisplay.GetComponent<Text> ().text
= "0" + LapTimeManager.MinuteCount + ".";
               } else {
                    MinuteDisplay.GetComponent<Text> ().text
= "" + LapTimeManager.MinuteCount + ".";
               }

               MilliDisplay.GetComponent<Text> ().text = "" +
LapTimeManager.MilliCount;
          }
          PlayerPrefs.SetInt ("MinSave",
LapTimeManager.MinuteCount);
          PlayerPrefs.SetInt ("SecSave",
LapTimeManager.SecondCount);
          PlayerPrefs.SetFloat ("MilliSave",
LapTimeManager.MilliCount);
```

```
          PlayerPrefs.SetFloat ("RawTime",
LapTimeManager.RawTime);
          LapTimeManager.MinuteCount = 0;
          LapTimeManager.SecondCount = 0;
          LapTimeManager.MilliCount = 0;
          LapTimeManager.RawTime = 0;
          LapCounter.GetComponent<Text> ().text = "" +
LapsDone;
          HalfLapTrig.SetActive (true);
          LapCompleteTrig.SetActive (false);
     }

}
```

Кожна деталь має свій колір і відповідне значення очок яке вона відає при колізії з нею, коли машина гравця проходить через одну з деталей, активується OnTriggerEnter цієї деталі та гравцю зачисляється відповідна кількість очок, на прикладі червоної деталі додається 100 до очок які гравець вже зібрав ModeScore.CurrentScore += 100 (Лістинг 7).

Лістинг *7 видача очок за колекціонування деталей різного кольору*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class BlueScore : MonoBehaviour {

    void OnTriggerEnter()
    {
        ModeScore.CurrentScore += 50;
        gameObject.SetActive(false);
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class RedScore : MonoBehaviour {

    void OnTriggerEnter()
    {
        ModeScore.CurrentScore += 100;
        gameObject.SetActive(false);
    }
}
using System.Collections;
using System.Collections.Generic;
```

```
using UnityEngine;
public class YellowScore : MonoBehaviour {

    void OnTriggerEnter()
    {
        ModeScore.CurrentScore += 25;
        gameObject.SetActive(false);
    }
}
```

Лістинг *8 ігровий таймер з точністю до мілісекунд*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class LapTimeManager : MonoBehaviour {
     public static int MinuteCount;
     public static int SecondCount;
     public static float MilliCount;
     public static string MilliDisplay;
     public GameObject MinuteBox;
     public GameObject SecondBox;
     public GameObject MilliBox;
     public static float RawTime;



     void Update () {
          MilliCount += Time.deltaTime * 10;
          RawTime += Time.deltaTime;
          MilliDisplay = MilliCount.ToString ("F0");
          MilliBox.GetComponent<Text> ().text = "" +
MilliDisplay;
          if (MilliCount >= 10) {
               MilliCount = 0;
               SecondCount += 1;
```

```
        }
        if (SecondCount <= 9) {
            SecondBox.GetComponent<Text> ().text = "0" +
SecondCount + ".";
        } else {
            SecondBox.GetComponent<Text> ().text = "" +
SecondCount + ".";
        }
        if (SecondCount >= 60) {
            SecondCount = 0;
            MinuteCount += 1;
        }
        if (MinuteCount <= 9) {
            MinuteBox.GetComponent<Text> ().text = "0" +
MinuteCount + ":";
        } else {
            MinuteBox.GetComponent<Text> ().text = "" +
MinuteCount + ":";
        }
    }
}
```

У лістингу 9 реалізовано зворотній таймер до початку гри, який очікує 3 секунди, лише після чого LapTimer.SetActive та CarControls.SetActive отримують параметри true та активують таймери і надають гравцю та штучному інтелекту можливість керувати машиною.

Лістинг *9 Зворотній таймер перед початком гонки*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class Countdown : MonoBehaviour {
    public GameObject CountDown;
```

```
    public AudioSource GetReady;
    public AudioSource GoAudio;
    public GameObject LapTimer;
    public GameObject CarControls;
    public AudioSource LevelMusic;
    void Start () {
        StartCoroutine (CountStart ());
    }
    IEnumerator CountStart () {
        yield return new WaitForSeconds (0.5f);
        CountDown.GetComponent<Text> ().text = "3";
        GetReady.Play ();
        CountDown.SetActive (true);
        yield return new WaitForSeconds (1);
        CountDown.SetActive (false);
        CountDown.GetComponent<Text> ().text = "2";
        GetReady.Play ();
        CountDown.SetActive (true);
        yield return new WaitForSeconds (1);
        CountDown.SetActive (false);
        CountDown.GetComponent<Text> ().text = "1";
        GetReady.Play ();
        CountDown.SetActive (true);
        yield return new WaitForSeconds (1);
        CountDown.SetActive (false);
        GoAudio.Play ();
        LevelMusic.Play ();
        LapTimer.SetActive (true);
        CarControls.SetActive (true);
    }


}
```

У лістингу 10 реалізована фінішна анімація за допомогою прокрутки камери навколо машини гравця який зайняв перше місце по результатам гонки через параметр transform.Rotate.

Лістинг *10 фінішна анімація*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class FinishRotate : MonoBehaviour {
    void Update () {
        transform.Rotate (0, 1, 0, Space.World);
    }
```

```
}
```

Рис.*12 Панель Score*

Рис.*13 Табло-таймер*

Опонентом у основному режимі гри виступає ШІ який реалізується за допомогою наданих інструментів Unity та штучного автомату якому задаються скрипти (Див Лістинг 13 — 15).

Пошук шляху по трасі у ШІ заданий наступним чином: вздовж траси розташовані невидимі маркери які активуються один за одним (Див. Лістинг 11). На початку гри активний маркер 1, інші маркери деактивовані, при колізії з маркером1 він деактивується та активує маркер2 і так далі, штучний інтелект шукає найкоротший шлях до активного маркеру, коли ШІ проходить маркер 7 та маркер трекер число маркеру скидається та ШІ знову шукає шлях до маркеру 1, траса з маркерами представлена на рис.14

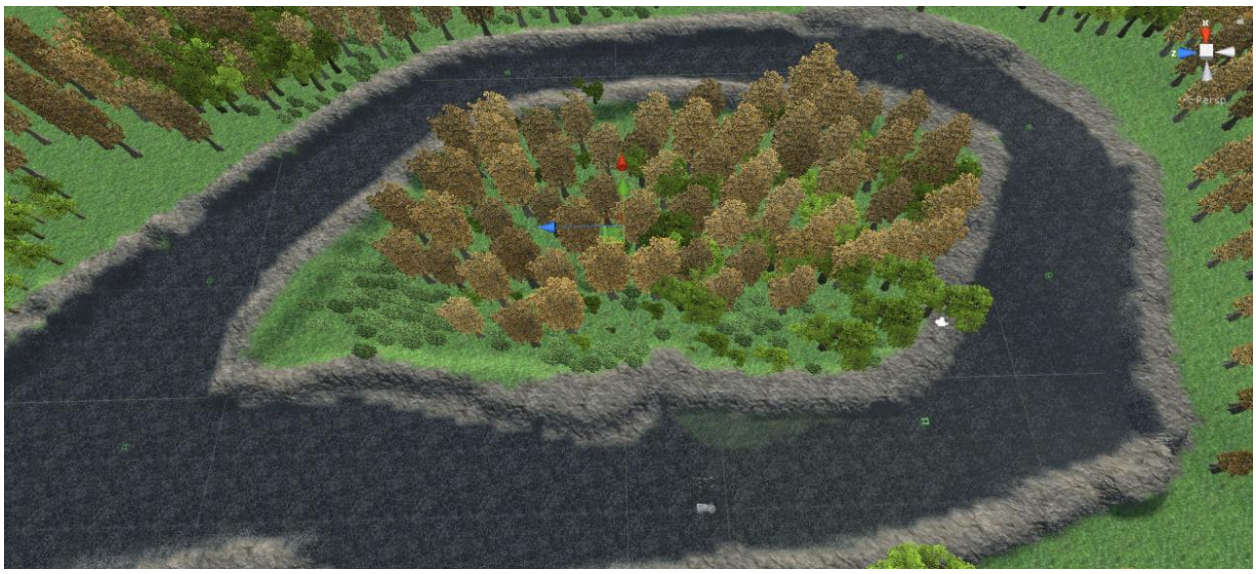

Рис.*14 Траса з маркерами для ШІ*

У лістингу 11 реалізовано активацію маркерів пошуку напрямку для ШІ, з початку гри активний Mark01, інші маркери є неактивними, за допомогою collision.gameObject.tag проходить перевірка на проходження ШІ маркеру після чого пройдений маркер деактивується та активує наступний BoxCollider.

Після того як ШІ проходить останній 7й маркер та фінішну лінію MarkTracker змінює параметр на 1 та ШІ знову шукає Mark01.

Лістинг *11 пошук ШІ шляху по маркерам*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Dreamcar01Track : MonoBehaviour {
     public GameObject TheMarker;
     public GameObject Mark01;
     public GameObject Mark02;
     public GameObject Mark03;
     public GameObject Mark04;
     public GameObject Mark05;
     public GameObject Mark06;
     public GameObject Mark07;
     public int MarkTracker;
     void Update () {
          if (MarkTracker == 0) {
```

```
            TheMarker.transform.position =
Mark01.transform.position;
        }
        if (MarkTracker == 1) {
            TheMarker.transform.position =
Mark02.transform.position;
        }
        if (MarkTracker == 2) {
            TheMarker.transform.position =
Mark03.transform.position;
        }
        if (MarkTracker == 3) {
            TheMarker.transform.position =
Mark04.transform.position;
        }
        if (MarkTracker == 4) {
            TheMarker.transform.position =
Mark05.transform.position;
        }
        if (MarkTracker == 5) {
            TheMarker.transform.position =
Mark06.transform.position;
        }
        if (MarkTracker == 6) {
            TheMarker.transform.position =
Mark07.transform.position;
        }
    }
    IEnumerator OnTriggerEnter(Collider collision) {
        if (collision.gameObject.tag == "Dreamcar01") {
            this.GetComponent<BoxCollider> ().enabled =
false;
            MarkTracker += 1;
            if (MarkTracker == 7) {
                MarkTracker = 0;
            }
            yield return new WaitForSeconds (1);
            this.GetComponent<BoxCollider> ().enabled =
true;
        }
    }
}
```

Лістинг *12 підключення наданого Unity ШІ для машини суперника*

```
var CarControl : GameObject;
var Dreamcar01 : GameObject;
function Start () {
```

```
    CarControl.GetComponent("CarController").enabled =
true;
    Dreamcar01.GetComponent("CarAIControl").enabled = true;
}
```

У грі змодельовано дві траси для кращого тестування моделі руху ШІ (див.рис.15-16).

Моделювання мап та текстури були взяті з безкоштовних джерел [8] та представлені у наданих двигуном можливостях, такі як редагування поверхні траси, створення дерев, тощо.

Рис.*15 Денна мапа*

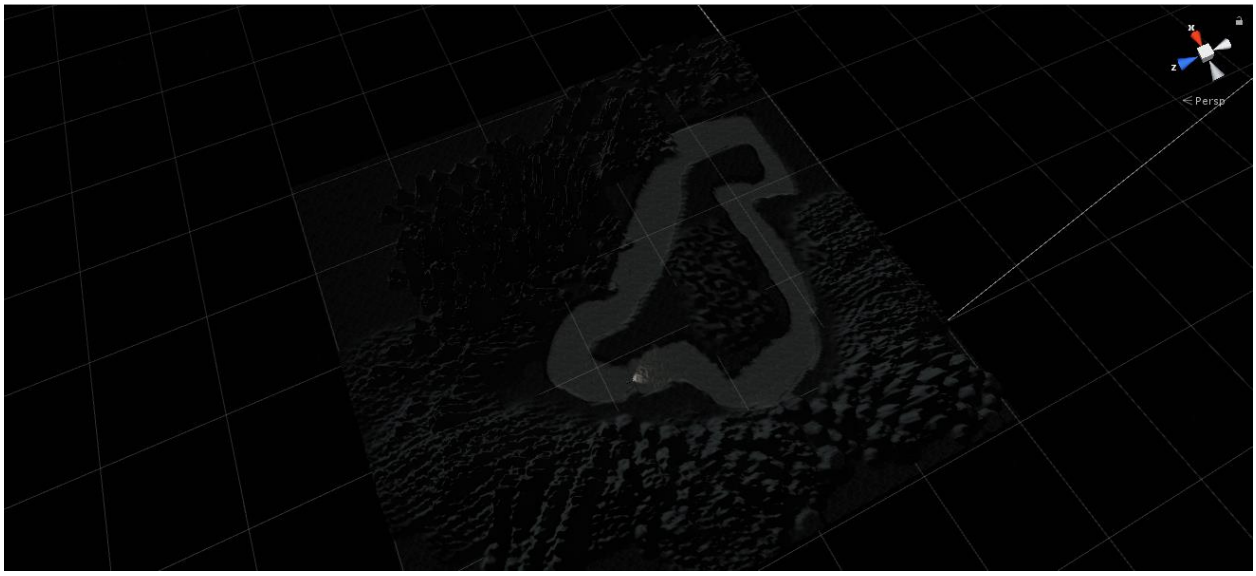

Рис.*16 Нічна мапа*

Модель ШІ була використана з наданих в Unity, сама модель включає в себе безліч можливостей для налаштування та надання їй більшої складності, реалістичності або схожості на людську поведінку. При імплементуванні у гру з більшою кількістю гравців ШІ може також реагувати на позиції суперників, колізію з ними та перешкодами, що надає йому змогу корегувати свій шлях, роблячи його оптимальним та найбільш вигідним в плані часу.

У наведених нижче лістингах ШІ приймає дані про оточуючу середу, наприклад:

Приведений лістинг відповідає за налаштування параметрів які використовуються для регулювання поведінки ШІ у плані керування, регулювання швидкості, використання гальм та реакцію на колізію з перешкодами або іншими гравцями.

CautiousMaxAngle, SteerSensitivity та BrakeCondition є змінними параметрами керування ШІ які регулюють швидкість, максимальний кут повороту на управління гальмами машини, зміна показників цих параметрів напряму впливає на поведінку ШІ та може змінити модель керування машиною, додаючи або зменшуючи складність опонента (Лістинг 13).

Лістинг *13 параметри та змінні для зміни налаштувань ШІ*

```
using System;
using UnityEngine;
using Random = UnityEngine.Random;

namespace UnityStandardAssets.Vehicles.Car
{
    [RequireComponent(typeof (CarController))]
    public class CarAIControl : MonoBehaviour
    {
        public enum BrakeCondition
        {
            NeverBrake,
            TargetDirectionDifference,
            TargetDistance,
        }
        [SerializeField] [Range(0, 1)] private float
m_CautiousSpeedFactor = 0.05f;
        [SerializeField] [Range(0, 180)] private float
m_CautiousMaxAngle = 50f;
        [SerializeField] private float
m_CautiousMaxDistance = 100f;
        [SerializeField] private float
m_CautiousAngularVelocityFactor = 30f;
        [SerializeField] private float m_SteerSensitivity =
0.05f;
        [SerializeField] private float m_AccelSensitivity =
0.04f;
        [SerializeField] private float m_BrakeSensitivity =
1f;
        [SerializeField] private float
m_LateralWanderDistance = 3f;
        [SerializeField] private float m_LateralWanderSpeed
= 0.1f;
        [SerializeField] [Range(0, 1)] private float
m_AccelWanderAmount = 0.1f;
        [SerializeField] private float m_AccelWanderSpeed =
0.1f;
        [SerializeField] private BrakeCondition
m_BrakeCondition = BrakeCondition.TargetDistance;
        [SerializeField] private bool m_Driving;
        [SerializeField] private Transform m_Target;
        [SerializeField] private bool
m_StopWhenTargetReached;
        [SerializeField] private float
m_ReachTargetThreshold = 2;

        private float m_RandomPerlin;
```

```
        private CarController m_CarController;
        private float m_AvoidOtherCarTime;
        private float m_AvoidOtherCarSlowdown;
        private float m_AvoidPathOffset;
        private Rigidbody m_Rigidbody;
```

Надання ШІ можливості використовувати гальма, перевіряти дистанцію до перешкод або суперників та регулювання їх використання на поворотах, при колізії, тощо (Лістинг 14).

Лістинг *14. Керування машиною ШІ*

```
private void FixedUpdate()
{
if (m_Target == null || !m_Driving)
{
m_CarController.Move(0, 0, -1f, 1f);
}
else
{
Vector3 fwd = transform.forward;
if (m_Rigidbody.velocity.magnitude >
m_CarController.MaxSpeed*0.1f)
{
fwd = m_Rigidbody.velocity;
}
float desiredSpeed = m_CarController.MaxSpeed;
switch (m_BrakeCondition)
{
case BrakeCondition.TargetDirectionDifference:
{
float approachingCornerAngle =
Vector3.Angle(m_Target.forward, fwd);
float spinningAngle =
m_Rigidbody.angularVelocity.magnitude*m_CautiousAngularVelo
cityFactor;
float cautiousnessRequired = Mathf.InverseLerp(0,
m_CautiousMaxAngle,
Mathf.Max(spinningAngle,
approachingCornerAngle));
desiredSpeed = Mathf.Lerp(m_CarController.MaxSpeed,
m_CarController.MaxSpeed*m_CautiousSpeedFactor,
cautiousnessRequired);
break;
}

case BrakeCondition.TargetDistance:
```

```
{
Vector3 delta = m_Target.position — transform.position;
float distanceCautiousFactor =
Mathf.InverseLerp(m_CautiousMaxDistance, 0,
delta.magnitude);
float spinningAngle=
m_Rigidbody.angularVelocity.magnitude*m_CautiousAngularVelo
cityFactor;

float cautiousnessRequired = Mathf.Max(
Mathf.InverseLerp(0, m_CautiousMaxAngle, spinningAngle),
distanceCautiousFactor);
desiredSpeed = Mathf.Lerp(m_CarController.MaxSpeed,
m_CarController.MaxSpeed*m_CautiousSpeedFactor,
cautiousnessRequired);
break;
}

case BrakeCondition.NeverBrake:
break;
```

Надання ШІ можливості приймати рішення щодо ухилення від можливих зіткнень з суперниками або перешкодами. Також ШІ може прийняти рішення не ухилятися а лише змінити швидкість. Основною причиною для таких дій є загроза застряти через зіткнення.

Float approachingCornerAngle=Vector3.Angle(m_Target.forward, fwd); перевірка кута нахилу нашої цілі порівняно з поточним напря-мком руху автомобіля також враховується поточний кут, на який ШІ повертає.

Автомобіль буде гальмувати, наближаючись до цілі, незалежно від її напрямку. Це корисно, якщо ви хочете, щоб машина прямувала до нерухомої цілі і зупинялася, коли прибуває до неї, приклад скрипту наведений нижче:

Vector3 delta = m_Target.position - transform.position; float distanceCautiousFactor = Mathf.InverseLerp(m_CautiousMaxDistance, 0, delta.magnitude);

Основним типом даних який використовується є Vector, він використовується для знаходження та оновлення позиції інших гравців та зміни реакцій на ці фактори.

Другим основним типом даних є float який використовується для збереження та зміни показників швидкості ШІ, гальмів та інших парамет-рів відповідаючих за керування машиною, детально зображено у Лістингу 15.

Вхідними даними є координати тих чи інших об`єктів та формули для вирахування векторів руху та зміни швидкості. Вихідними даними є float з параметрами які змінюють швидкість та вектори руху ШІ.

Рішення про зміну маршруту до наступної точки, маневру або зміни швидкості змінюються залежно від вхідних даних (Лістинг 15).

Лістинг *15 Реакції ШІ на оточення*

```
Vector3 offsetTargetPos = m_Target.position;
if (Time.time < m_AvoidOtherCarTime)
{
desiredSpeed *= m_AvoidOtherCarSlowdown;
offsetTargetPos += m_Target.right*m_AvoidPathOffset;
}
else
{
offsetTargetPos += m_Target.right*
(Mathf.PerlinNoise(Time.time*m_LateralWanderSpeed,
m_RandomPerlin)*2 — 1)*
m_LateralWanderDistance;
}
float accelBrakeSensitivity = (desiredSpeed <
m_CarController.CurrentSpeed)
? m_BrakeSensitivity
: m_AccelSensitivity;
float accel = Mathf.Clamp((desiredSpeed —
m_CarController.CurrentSpeed)*accelBrakeSensitivity, -1,
1);
accel *= (1 — m_AccelWanderAmount) +
(Mathf.PerlinNoise(Time.time*m_AccelWanderSpeed,
m_RandomPerlin)*m_AccelWanderAmount);
Vector3 localTarget =
transform.InverseTransformPoint(offsetTargetPos);
float targetAngle = Mathf.Atan2(localTarget.x,
localTarget.z)*Mathf.Rad2Deg;
float steer = Mathf.Clamp(targetAngle*m_SteerSensitivity, -
1, 1)*Mathf.Sign(m_CarController.CurrentSpeed);
m_CarController.Move(steer, accel, accel, 0f);
if (m_StopWhenTargetReached && localTarget.magnitude <
m_ReachTargetThreshold)
{
```

```
m_Driving = false;
}
}
}


private void OnCollisionStay(Collision col)
{
if (col.rigidbody != null)
{
var otherAI = col.rigidbody.GetComponent<CarAIControl>();
if (otherAI != null)
{
m_AvoidOtherCarTime = Time.time + 1;

if (Vector3.Angle(transform.forward,
otherAI.transform.position — transform.position) < 90)
{
m_AvoidOtherCarSlowdown = 0.5f;
}
else
{
m_AvoidOtherCarSlowdown = 1;
}

var otherCarLocalDelta =
transform.InverseTransformPoint(otherAI.transform.position)
;
float otherCarAngle = Mathf.Atan2(otherCarLocalDelta.x,
otherCarLocalDelta.z);
m_AvoidPathOffset = m_LateralWanderDistance*-
Mathf.Sign(otherCarAngle);
}
}
}


public void SetTarget(Transform target)
{
m_Target = target;
m_Driving = true;
}
}
}
```

3.6 Тестування ігрового застосунку

Тестування ШІ проводилося в процесі тестових матчів на різних мапах з розташованими на них маркерами пошуку шляху (Рис.15, 16).

У ході тестування змінювались невидимі стіни мапи та сама траса для перевірки реакції ШІ на зміну оточуючого середовища, у результаті тестування ШІ показав непогані результати, але також довів те, що для нормального функціонування ШІ у якості суперника йому потрібна велика кількість машинного навчання та більша кількість змінних перешкод для перевірки.

Незважаючи на неідеальну поведінку ШІ, навіть на цьому етапі він може буди сильним суперником та створити враження реального конкуруючого суперника.

Приклад роботи режиму Race Mode на денній мапі (Рис.17).

Старт гонки після зворотнього відліку:

Рис.*17 Старт гонки*

Продовження гонки у центрі мапи, ШІ має виграшну позицію, знаходиться попереду гравця:

Рис.*18 Продовження гонки з перемагаючим ШІ*

Кінець першого кола, продовження гонки, ШІ знаходиться на першій позиції:

Рис.*19 Фініш першого кола*

Перемога гравця та фінішна анімація у кінці:

Рис.*20 Фініш гонки*

ВИСНОВКИ

1.Виконано аналіз літературних джерел з теми розробки гоночних ігор на платформі Unity. Отримані дані дозволили поглибити розуміння основних принципів розробки ігор та дослідити сучасні тенденції у цій галузі.

2.Розглянуто предметну область гоночних ігор та вивчено основні аспекти геймплею, фізики руху транспортних засобів та логіку поведінки суперників.

3.Виконано аналіз різних засобів розробки гоночних ігор, зокрема фреймворку Unity. Проаналізовано їх переваги та недоліки, функціонал та можливості для реалізації геймплею.

4.Обрано Unity як основний засіб розробки гоночної гри. Цей вибір обгрунтовується широким функціоналом, підтримкою від спільноти розробників, можливістю створення реалістичної графіки та реалізацією фізики руху транспортних засобів.

5. Розроблено гоночну гру на платформі Unity, де суперником виступає штучний інтелект, що шукає шлях за допомогою невидимих маркерів.Штучний інтелект було створено за допомогою скриптів реагування на зовнішнє середовище та маркери пошуку. Реалізовано систему поведінки суперника, його реакцію на дії гравця та взаємодію з оточуючим середовищем.

6.Перевагами розробленої гри є реалістична модель фізики руху транспортних засобів, налаштований штучний інтелект суперника та захоплюючий геймплей. Проте, є певні обмеження, зокрема в області графічних можливостей та оптимізації продуктивності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Dorahn J.P. Unity Artificial Intelligence Programming: Add powerful, believable, and fun AI entities in your Unity projects. 2022. 308p.

2. Kim R. Unity AI Game Programming. Second Edition. 2015. 232p.

3. Palacios J. Unity Artificial Intelligence Cookbook. 2018. 334p.

4. Hocking J. Unity in Action: Multiplatform Game Development in C# 2018 400p.

5. UNITY MACHINE LEARNING AGENTS URL: https://unity.com/products/machine-learning-agents (дата звернення: 08.06.2023)

6. Need for Speed: Rivals Review on GameSpot URL: https://www.gamespot.com/reviews/need-for-speed-rivals-review/1900-6415555/ (дата звернення: 10.06.2023)

7. Project Cars 2 Review URL: https://www.gamespot.com/reviews/project-cars-2/1900-6416766/ (дата звернення: 10.06.2023)

8. Forza Motorsport 7 Review URL: https://www.gamespot.com/reviews/forza-motorsport-7-review/1900-6416781/ (дата звернення: 10.06.2023)

9. Free assets, music and etc. URL: https://jvunity.weebly.com/ (дата звернення: 08.06.2023)

10. Unreal engine URL: https://www.unrealengine.com/en-US/ (дата звернення: 09.06.2023)

11. GameMaker URL: https://gamemaker.io (дата звернення: 09.06.2023)