

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ПРОТОТИПУ 3D ГРИ НА ANDROID З
ВИКОРИСТАННЯМ МОВИ ПРОГРАМУВАННЯ C#
ТА СЕРЕДОВИЩА UNITY
DEVELOPMENT OF A 3D ANDROID GAME
PROTOTYPE USING C# AND UNITY

Виконав: студент(ка) 4 курсу, групи КНТ-128
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Шутко А.С.

(прізвище та ініціали)

Керівник Каплієнко Т.І.

(прізвище та ініціали)

Рецензент Шитікова О.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
« » 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Шутко Анни Сергіївни

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка прототипу 3D гри на Android з використанням мови програмування C# та середовища Unity. Development of a 3D Android Game Prototype Using C# and Unity

керівник проєкту (роботи) Каплієнко Тетяна Ігорівна, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від «27» квітня 2022 року № 102

2. Строк подання студентом проєкту (роботи) 7 червня 2022 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Вибір програмного забезпечення та його обґрунтування. 3. Основні рішення щодо реалізації компонентів гри. 4. Керівництво програміста. 5. Керівництво користувача. 6. Тестування і результати. 7. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-6 Основна частина	Каплієнко Т.І., доцент		
Нормоконтроль	Камінська Ж.К., асист.		

7. Дата видачі завдання “ 7 ” лютого 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2 тиждень	Розділ 1
3	Вибір програмного забезпечення та його обґрунтування	2-3 тижні	Розділ 2
4	Основні рішення щодо реалізації компонентів гри	3-4 тижні	Розділ 3
5	Розробка архітектури гри	4 тиждень	Розділ 4
6	Розробка програми	5-6 тижні	Розділ 5
7	Тестування та експериментальне дослідження програми	6 тиждень	Розділ 6
8	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	7 тиждень	Додатки
9	Захист роботи	8 тиждень	

Студент(ка)

(підпис) Шутко А.С.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Каплієнко Т.І.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
137 с., 42 табл., 72 рис., 2 дод., 42 джерела.

ІНТЕРАКТИВНА ГРА, КОГНІТИВНІ ФУНКЦІЇ, МОБІЛЬНА ГРА,
ANDROID, UNITY 3D GAME.

Об'єкт дослідження – процес розроблення відеоігор для розвитку когнітивних функцій.

Предмет дослідження – розробка прототипу 3D гри на Android з використанням мови програмування C# та середовища Unity.

Мета роботи – створення прототипу відеогри для розвитку деяких когнітивних навичок.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, пристрої Mi Pad 4, Redmi 9, Poco F3 та Redmi Note 7 Pro, міжплатформне середовище розробки Unity, інтегроване середовище розробки Visual Studio, мова програмування C#, персональний комп'ютер з процесором AMD Ryzen 5 під управлінням операційної системи Microsoft Windows 10.

Результати. Створено прототип 3D гри на Android, керований пристроями Mi Pad 4, Redmi 9, Poco F3 та Redmi Note 7 Pro, який у формі тривимірної інтерактивної гри взаємодіє з користувачем та стимулює його до розвитку швидкості реакції, уваги, логічного мислення.

Висновки. Розроблено прототип гри для дітей дошкільного віку, що вирішує поставлені задачі.

Галузь використання – розвиток когнітивних функцій: швидкість реакції, увага та логічне мислення.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 137 pages, 72 figures, 42 tables, 2 appendixes, 42 sources.

INTERACTIVE GAME, COGNITIVE FUNCTIONS, MOBILE GAME, ANDROID, UNITY 3D GAME.

Object of study is the process of developing video games for the development of cognitive functions.

Subject of study is development of a 3D Android game prototype using C# and Unity.

The purpose of the work is to create a prototype video game to develop some cognitive skills.

Materials, methods, and tools: structured and object-oriented programming, Mi Pad 4, Redmi 9, Poco F3 and Redmi Note 7 Pro devices, cross-platform development environment Unity, integrated development environments Visual Studio, programming language C#, personal computer with an AMD Ryzen 5 processor under the control of the Microsoft Windows 10 operating system.

Results. A prototype 3D game for Android is created which powered by the Mi Pad 4, the Redmi 9, the Poco F3 and the Redmi Note 7 Pro devices, which interacts with the user in the form of a three-dimensional interactive game and motivates it to develop reaction speed, attention, logical thinking.

Conclusions. A prototype game for preschool children is developed that solves the set tasks.

The field of use is a development of cognitive functions: reaction speed, attention and logical thinking.

ЗМІСТ

	С.
ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК.....	8
ВСТУП.....	9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	10
1.1 Актуальність проблеми	10
1.2 Інформація о роботі головного мозку	15
1.3 Поняття когнітивного розвитку.....	16
1.4 Дослідження відеоігор з точки зору когнітивного розвитку	19
1.5 Зв'язок жанрів ігор із різними когнітивними функціями.....	23
1.6 Модуючі фактори	29
1.7 Огляд існуючих аналогів.....	32
1.8 Аналіз цільової аудиторії	35
1.9 Постановка задач роботи.....	36
1.10 Висновок за першим розділом.....	36
2 ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ЙОГО ОБҐРУНТУВАННЯ...38	
2.1 Вибір ігрового рушія	38
2.2 Вибір мови програмування	43
2.3 Вибір середовища розробки.....	48
2.4 Висновок за другим розділом	51
3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ ГРИ.....52	
3.1 Аналіз структурної схеми гри.....	52
3.2 Проєктування локацій гри.....	56
3.3 Опис модулів мобільної гри.....	57
3.4 Алгоритми розробленої гри	67
3.5 Висновок за третім розділом.....	77
4 КЕРІВНИЦТВО ПРОГРАМІСТА.....78	
4.1 Призначення й умови застосування гри	78
4.2 Основні вимоги до гри.....	78
4.3 Характеристики гри	79

4.4 Звертання до гри.....	80
4.5 Початкові та вихідні дані	80
5 КЕРІВНИЦТВО КОРИСТУВАЧА.....	82
5.1 Призначення гри.....	82
5.2 Умови виконання гри.....	82
5.3 Виконання гри	83
6 ТЕСТУВАННЯ І РЕЗУЛЬТАТИ.....	86
6.1 Опис методики проведення тестування.....	86
6.2 Результати використання розробленої гри.....	86
6.3 Оптимізація гри	93
6.4 Висновки за розділом 6	100
ВИСНОВКИ	101
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	102
ДОДАТОК А Текст програми	106
ДОДАТОК Б Слайди презентації.....	127

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

AVG	– Action Video Game;
CIL	– Common Intermediate Language;
FG	– Fantasy Game;
FPS	– First-Person Shooters;
FPS	– Frames Per Second
IDE	– Integrated Development Environment;
PS	– PlayStation;
RPG	– Role-Playing Game;
RTS	– Real-Time Strategy;
SG	– Simulation Game;
SVG	– Strategy Video Game;
TBS	– Turn-Based Strategy;
TG	– Traditional Game;
TPG	– Third-Person Games;
UE	– Unreal Engine;
VS	– Visual Studio;
OC	– операційна система;
ПК	– персональний комп'ютер;
ЦНС	– центральна нервова система.

ВСТУП

За останнє десятиліття відеоігри проникли у велику кількість сфер нашого життя. При знайомстві ближче зі світом відеоігор можна помітити, що прискіпливо обрані ігри будуть дуже зручним інструментом для навчання. Відеоігри поділяються за призначенням і цілям створення ігри на розважальні ігри та створені з метою навчання, тобто спеціальні програми в ігровій формі, що подають конкретний навчальний матеріал.

Також відеоігри застосовуються для розвитку когнітивних навичок людини. Когнітивні навички включають наступне: постійна увага, виборча увага, розділена увага, довгострокова пам'ять, робоча пам'ять, логіка і міркування, слухова обробка, візуальна обробка і швидкість обробки. Ці навички використовуються в усіх повсякденних завданнях, таких як відповідь на телефонний дзвінок, відповідь на повідомлення друга або навіть перегляд телевізора [1].

Крім того, ігри можуть допомогти в підготовці фахівців, чия робота пов'язана з підвищеною візуальною увагою – наприклад, хірургів, правоохоронців або військових. Останнє, до речі, підтверджується навіть на дуже високому рівні. Здатність швидко засвоювати інформацію, реагувати і координувати дії, залишаючись при цьому спокійним в умовах тиску, часто є ознакою людей, які добре грають в шутери. Але тривалість гри повинно бути обмежено, якщо час на відеоігри витісняє час на інші завдання, інші навички можуть з часом постраждати [2].

Виходячи з цього, робота спрямована на розробку відеоігри, основною метою якої є розвиток деяких когнітивних навичок людини.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність проблеми

У ХХІ столітті відеоігри міцно займають одну з найвищих позицій у сфері розваг і хобі. У зв'язку з їх високою популярністю учені все частіше звертають увагу на тему "Як відеоігри впливають на мозок людини", їхній вплив на когнітивний розвиток людини: на пам'ять, увагу, мислення та інтелектуальні функції загалом [3].

Ігри – це не лише дитячі розваги. Фактично, навіть літні люди дістають користь від розваг. Ігри для літніх людей покращують когнітивні навички, стимулюють мозок для відвертання деменції і забезпечують соціалізацію для здорового благополуччя.

З віком загальний об'єм мозку зменшується, особливо кількість білої речовини. Це робить літніх людей схильними до втрати пам'яті, а також до зниження інших когнітивних навичок, таких як увага та реакція. Ці тонкі зміни білої речовини вважаються нормальними, але відеоігри можуть покращити когнітивні навички [4].

1.1.1 Поняття відеоігри та її ознаки

Відеоігри вперше побачили світ наприкінці 50-х років і з того часу вони продовжували вдосконалюватися, покращуючи якість графіки, ігрову механіку та маркетинг [5].

Відеогра – гра, в яку користувачі грають завдяки аудіовізуальному обладнанню, що може бути заснована на будь-якій історії.

Існує 4 ключові ознаки, якими характеризується сучасна відеогра і які можуть бути розглянуті окремо і як єдине явище:

- гра;
- ігрова взаємодія;
- аудіовізуальне обладнання;

– історія.

Відеогра відноситься до типу діяльності, що має певний набір правил, кінцеву мету (перемога) та процес досягнення цієї мети [6].

1.1.2 Ігрові платформи та їх порівняння

До популярних ігрових платформ відносяться: комп'ютери, приставки та смартфони.

Одна з популярних платформ – персональний комп'ютер (ПК).

Статистично порівнюючи з консолями, вартість ігор на ПК у 2-3 рази дешевша, ніж за ідентичну копію на тій же PlayStation або Xbox. Також ПК мають можливість модифікації. Консолі випускаються на тривалий період використання без можливості покращення, приблизно від 6 до 8 років. Це встановлює поріг продуктивності, який обмежує ігри. ПК ж має можливість апдейта, на ньому можна модульно замінити: оперативну пам'ять, процесор, відеокарту, а також інші види комплектуючих. Ігри з кожним роком стають все вимогливішими до "заліза", тому геймерам, які прагнуть максимальної потужності доводиться кожні кілька років купувати дорогі комплектуючі. У той же час на консолях "залізо" не змінюється роками, а якість графіки приблизно відповідає середнім-високим налаштуванням на ПК, ще й без просідань FPS. Ціни приставок варіюють у районі найдорожчих відеокарт. Також є безліч маніпуляторів, які недоступні для приставок: клавіатури, миші, кермо, геймпади, джойстики та інші. Існує багато ексклюзивів. Багато проєктів виходять виключно на комп'ютери: Total War, XCOM 2, S.T.A.L.K.E.R., World of Warcraft та інші [7].

Ігрова приставка від Microsoft також дуже популярна, зважаючи на відмінні ексклюзиви та відносну дешевизну. Таких приставок на даний момент випущено лише три: Xbox One, Xbox 360 і Xbox. Остання версія – Xbox One сильно відстає від основної конкуруючої консолі від Sony. Відрізняється ця приставка своїм ігровим маніпулятором Kinect, який дозволяє взаємодіяти з

консоллю не використовуючи аксесуари лише жестами і голосом. На основі цієї технології було розроблено безліч цікавих проектів, якими можна розважатися у колі друзів та близьких. Багато розробників роблять пріоритет у розробці ігор на консолях, тому що продажі на них набагато вищі. Причина всьому – піратство. Звичайно, багато виробників намагаються створювати нові види захисту, проте далеко не завжди вони виходять дієвими [7].

Найвизнаніша приставка, яка дещо дорожча, остання версія консолі, яка вийшла з-під крила Sony, – це PlayStation(PS) 4. Вона є останньою версією гілки PS, яка вже продася в 40 мільйонів копій, обминаючи при цьому конкуруючу Xbox One майже в 2 рази. Більше того, "залізо" PS 4 значно потужніше, отже на ній краща якість картинки, ніж на тій же Xbox One. PS4 має безліч визнаних ексклюзивів, серед яких: The Last of Us, Uncharted, God of War, Killzone, та багато інших. Серед усіх ігрових консолей геймери по всьому світу віддають перевагу саме PS [7].

У порівнянні з консолями та ПК, смартфони на базі ігрової індустрії на Andorid та IOS тільки починають набирати популярність, тому безліч розробників все більше кладуть око на саме мобільні платформи. Мобільні магазини вже поповнилися безліччю популярних серій ігор, таких як відома серія ігор – GTA, серія Modern Combat, Need for Speed та багатьма іншими відомими тайтлами. У минулому мобільні телефони не мали змоги запускати щось складніше за "Змійку". Тепер вони мають гарну картинку і звук, що дозволяє їм навіть конкурувати з ПК і консолями. Останніми роками ігрові платформи виходять за межі стаціонарних пристроїв, за типом комп'ютерів або приставок. Операційні системи (ОС) Android та iOS нині встановлені на більшості смартфонів, доступних на ринку. Є також Windows Phone, проте ігор на ньому виходить значно менше. Стосовно основних конкурентів, Android зараз має найвищу частку ринку – 80% проти 15% у iOS. Остання, у свою чергу, встановлюється лише на девайсах Apple [7].

Комп'ютерні ігри були домінуючими протягом певного часу, але недавня тенденція призвела до того, що консольні геймери зайняли ще більшу частку

ринку, тоді як індустрія мобільних ігор домінує у гонці платформ. Мобільні ігри – найпопулярніший спосіб грати серед геймерів, за ними йдуть консолі, а далі ПК [8].

Порівняння ігрових платформ подано в табл. 1.1.

Таблиця 1.1 — Порівняння ігрових платформ

Властивості	Ігрові платформи				
	ПК	Xbox	PlayStation	iOS	Android
Прибутки	Низькі	Середні	Середні	Високі	Високі
Бюджет	Високий	Середній	Середній	Високий	Низький
Частота гри	Низька	Середня	Середня	Висока	Висока
Ексклюзиви	Присутні	Відсутні	Присутні	Присутні	Присутні
Портативність	Відсутня	Відсутня	Відсутня	Присутня	Присутня
Багато-функціональність	Висока	Низька	Низька	Середня	Середня

Оцінки встановлені згідно коментарям та оцінкам користувачів ігор [8].

Індустрія мобільних ігор є найпопулярнішою з точки зору частки ринку, при цьому консольні ігри посідають друге місце, а ігри для ПК – останнє, тоді як індустрія мобільних ігор домінує в гонці платформ і зайняла перше місце. Android на даний момент має найвищу частку ринку – 80% проти 15% у iOS. Остання, у свою чергу, встановлюється лише на девайсах Apple. Для смартфонів на Android та iOS дохід на останній рік становив – 130 млрд. доларів, на ПК – 65 млрд. доларів, а на консолі – 25 млрд. доларів.

Бюджет. Один із головних критеріїв вибору ігрової платформи – кількість грошей, яку користувачі готові витратити на неї. Найдорожчим у придбанні буде ПК, далі консолі і iOS та найдешевші смартфони на Android.

Найчастіше мобільні геймери грають на своїх смартфонах, особливо в екшени та шутери; 57% геймерів у США грають на смартфонах. Тим не менш, багато користувачів також грають на планшетах, граючи в казуальні ігри, які,

до речі, подобаються 71% мобільних геймерів. 46% гравців у США – консольники. Зазвичай вони грають в екшн-ігри (60%) або шутери (57%), а казуальні ігри не такі популярні серед консольних геймерів. Консолі простіше налаштовувати та використовувати, вони часто більш доступні за ціною, і в них є такі функції, як допомога у прицілюванні у шутерах. Близько 42% геймерів у США грають на ПК, а 71% з них більшу частину часу грають у казуальні ігри.

Ексклюзиви. Багато проєктів виходять виключно на комп'ютери: World of Warcraft, Total War, S.T.A.L.K.E.R., XCOM 2 та інші. Для Xbox немає ексклюзивів. З додаванням підписки "Xbox Game Pass" всі ексклюзиви для платформи Xbox стають ексклюзивами Microsoft, тобто їх можна запускати як на Xbox, так і на ПК, що по суті поєднує всі ексклюзиви для Xbox з Windows, проте не всі ігри для ПК підтримують ігрову платформу Xbox, отже ПК – має ексклюзиви, а Xbox – ні. Що стосується ексклюзивів для PlayStation, на даний момент найпопулярнішими є такі проєкти: серії Gran Turismo, Uncharted, Killzone, God of War, infamous, а також одиночні ігри Heavy Rain, The Last of Us і The Order 1886. Для Android і iOS є ексклюзиви як для кожної ОС, так і для обох. До них можна віднести: Clash Of Clans та Clash Royale.

Портативність. Можливість грати де і коли завгодно. PlayStation 4, Xbox One та ПК не можна назвати портативними, а ось мобільні – можна. Мобільні ігри знаходяться на постійному підйомі, і багато користувачів вважають за краще грати в онлайн-ігри, не встаючи зі свого дивана, а не сидячи за комп'ютером.

Якщо користувачу потрібна реальна функціональність, то краще купувати ПК – це універсальні машини, на яких можна робити все, що завгодно: писати музику, розробляти програми, створювати документи, виходити інтернет, редагувати зображення. Менш функціональними є Android або iOS, але там теж є свої переваги та функціонал. Консолі створені переважно для ігор, і крім цього вони вміють не так вже й багато: відтворювати музику, показувати фільми та серіали [8].

Після порівняння було обрано розроблювати гру на Android. Бо це найпопулярніша та найдешевша платформа. Є можливість грати в гру усюди, як дома, в громадських місцях, або на вулиці.

1.2 Інформація о роботі головного мозку

Мозок – це складний орган, який є частиною центральної нервової системи (ЦНС). Він не є м'язом. Цей орган складається з мільйонів нейронів, пов'язаних між собою. Кожен окремо і всі разом вони регулюють функції нашого мозку і тіла [9].

Як головний орган ЦНС, він контролює та регулює більшість функцій людського організму. Починаючи від життєво важливих, таких, як, наприклад, дихання або серцеві ритми, сон, голод, спрага до вищих функцій: міркування, пам'ять, увага, контроль емоцій і поведінка. Мозок регулює всі вироблені нами уві сні і наяву дії: коли ми дихаємо або ковтаємо, дивимося, слухаємо, щось чіпаємо або пробуємо, читаємо або пишемо, співаємо або танцюємо, думаємо в тиші або говоримо, любимо або ненавидимо, ходимо або бігаємо, плануємо чи діємо спонтанно, уявляємо чи створюємо [10].

Складемо список основних функцій мозку:

- контроль життєвих функцій: контроль температури тіла, артеріального тиску, частота серцевих скорочень, дихання, сон, харчування;
- прийом, обробка, інтеграція та інтерпретація всієї отриманої від органів чуття інформації: зір, слух, смак, дотик і нюх;
- контроль рухів та поз тіла: рух пішки, біг, мова, стояння на місці;
- відповідає за наші емоції та поведінку;
- дозволяє нам думати, міркувати, відчувати, бути;
- контролює вищі когнітивні функції: пам'ять, здатність до навчання, сприйняття, виконавчі функції [11]-[12].

Тобто, мозок відповідає за інтегрування всієї інформації, що надходить від органів чуття, і за організацію відповіді на неї. Контролює емоції, моторні та

всі вищі когнітивні функції: міркування, емоційний вираз, пам'ять [13], навчання.

Також у мозку знаходяться сіра речовина та біла речовина. Тканина білої речовини є будинком для спеціалізованих клітин, що називаються нейронами, і нервових волокон, відомих як аксони. Аксони інкапсульовані мієліном – жировою оболонкою, яка полегшує обмін хімічними повідомленнями по нервовій системі. Навчання і різні інші функції мозку можливі, тому що біла речовина забезпечує зв'язок між різними областями мозку [4].

Головний мозок працює шляхом передачі інформації між нейронами за допомогою електрохімічних імпульсів.

Нейрони – клітини, що відповідають за отримання, обробку та передачу інформації, як на внутрішньоклітинному, так і міжклітинному рівні. Роблять вони це за допомогою електрохімічних сигналів (нервових імпульсів), які називаються потенціалом дії нейрона.

Передача інформації між нейронами триває мілісекунди. Одночасно і скоординовано відбуваються сотні з'єднань, які дозволяють нам адекватно сприймати інформацію, розуміти її і відповідати навколишньому світу [9].

1.3 Поняття когнітивного розвитку

Завдяки нейронній пластичності, а також здатності встановлювати нові нейронні зв'язки і посилювати старі, людина може поліпшити свої когнітивні навички [9]. Когнітивний розвиток, також відомий як інтелектуальний розвиток, – цей неврологічний і психологічний розвиток різних функцій мислення. Це спричиняє за собою застосування когнітивних навичок для свідомої інтерпретації довкілля. Когнітивний розвиток – це тема наукових досліджень в таких областях, як психологія і нейробіологія [14].

1.3.1 Области когнитивного розвитку

Когнітивний розвиток включає основні компоненти мислення. Для оптимального когнітивного розвитку краще всього зосередитися на областях обробки інформації, інтелекту, мислення, пам'яті і мови.

Обробка інформації – когнітивне сприйняття дозволяє нам організувати і зрозуміти світ, що оточує нас, через стимули, які ми отримуємо від органів чуття. Ми отримуємо інформацію за допомогою п'яти основних почуттів, таких як зір, слух, смак, нюх і дотик, а також через інші, менш відомі, як, наприклад, пропріоцепція (відчуття положення частин власного тіла в просторі і один відносно одного) або інтероцепція (процес сприйняття нервовою системою збудження внутрішніх органів, що наприклад, дає нам можливість переживати почуття спраги або голоду). Отримавши усю цю інформацію, мозок створює нове знання.

Пам'ять – це когнітивна функція, що дозволяє кодувати, зберігати і відновлювати отриману раніше інформацію в міру необхідності. Пам'ять є базовим процесом в навчанні, а також формує нашу ідентичність. Існує безліч видів і класифікацій пам'яті, зокрема, короткочасна пам'ять, здатність тимчасово утримувати інформацію в голові (наприклад, запам'ятати номер телефону до того моменту, поки людина його не запише на листку паперу), і довготривала пам'ять, тобто знання і спогади, які людина зберігає довго. Довготривала пам'ять, у свою чергу, підрозділяється на декларативну, яка включає як знання, придбані за допомогою мови і освіти (наприклад, знання про те, що громадянська війна в Іспанії закінчилася в 1939 році), так і знання, отримані через особистий життєвий досвід (наприклад, спогади про блюда, які готувала бабуся в селі); а також на процедурну пам'ять – пам'ять на дії (наприклад, водіння автомобіля або їзда на велосипеді). Існують також слухова пам'ять, контекстуальна пам'ять, пам'ять на імена, пізнавання.

Мова – це здатність висловлювати наші почуття і думки. За допомогою цього інструменту люди спілкуються, збирають та передають інформацію про

себе і навколишній світ. Мова і мислення розвиваються паралельно, тісно пов'язані між собою і значно впливають один на одного.

Увага – це когнітивний процес, який дозволяє усім сконцентруватися на певному стимулі або виді діяльності для того, щоб глибше усвідомити його. Увага є найважливішою функцією, що потрібна людям постійно і щодня. Більше того, вважається, що саме за допомогою уваги регулюються інші когнітивні процеси: від сприйняття (увага потрібна для розпізнавання стимулів, що поступають від органів чуття) до навчання і складних міркувань.

Мислення є найважливішим когнітивним процесом. Дозволяє об'єднувати усю отриману інформацію і встановлювати взаємозв'язок між складовими її даними. Для цього використовуються міркування, синтез і рішення завдань, тобто виконавчі функції.

Навчання – процес, що дозволяє додати нову інформацію до існуючих знань. Під навчанням мається на увазі різні його види, починаючи від навчання поведінці (як чистити зуби або ходити) до усіх тих знань, які отримуються в школі і при соціалізації. Дослідники визначають когнітивне навчання як процес, при якому інформація поступає в когнітивну систему і міняє її.

Когнітивні процеси можуть бути натуральними і штучними, свідомими або неусвідомлюваними, але, в основному, вони настільки швидкі і так часто відбуваються, що ми їх навіть не усвідомлюємо. Наприклад, коли хтось йде по вулиці і бачить червоне світло світлофора, він запускає когнітивний процес, в результаті якого приймає рішення (переходити дорогу або ні). Насамперед він зосереджує увагу на світлофорі і за допомогою зору бачить, що горить червоний колір. За долі секунд людина згадує, що на червоне світло світлофора дорогу переходити не можна, але також згадує і про те, що іноді люди все одно це роблять якщо немає машин. Тут, ймовірно, буде прийнято перше рішення: дочекатися доки світлофор поміняє колір або подивитися по сторонах (перенаправити увагу), щоб перевірити, чи їдуть автомобілі і чи безпечно перейти дорогу [14].

1.4 Дослідження відеоігор з точки зору когнітивного розвитку

Велика група досліджень відеоігор присвячена вивченню їх впливу на когнітивний розвиток людини. Усупереч поширеній думці, цей вид розваги має більше позитивних ефектів, ніж негативних. Існують суперечливі дослідження. Загальне уявлення про те, що від ігор мозок псується, знаходиться на рівні суспільної свідомості і не зовсім узгоджується з даними досліджень [16].

Вивчення відеоігор з точки зору когнітивного розвитку починається дуже рано. Перші експериментальні роботи відносяться до 1980-1990-м рокам і пов'язані з вивченням просторових здібностей. Просторове мислення (*spatial thinking*) – це ціла група здібностей людини, пов'язаних з орієнтацією в тривимірних просторах, в перенесенні тривимірних просторів на двомірні. Наприклад, коли людині потрібно зорієнтуватися по карті: карта плоска, а місто, що оточує, цілком собі тривимірне.

Дослідження показують, що просторові здібності у гравців в відеоігри розвиваються. Саме з іграми частенько зв'язують зменшення гендерних відмінностей, що відбувається нині, в просторових здібностях між чоловіками і жінками. Вважається, що жінки гірше орієнтуються в просторі, ніж чоловіки. З цим пов'язана маса анекдотів про те, як жінки крутять карту і водять машину. Останнім часом у зв'язку з розвитком технологій ці відмінності практично стираються. Дівчата, що грають в відеоігри, так само добре справляються з просторовими завданнями, як і чоловіки, що грають в відеоігри. І усі геймери роблять це краще, ніж люди, які не грають в ігри [16]-[17].

Проте багато досліджень побудовані на порівнянні людей, які багато грають в відеоігри, і людей, які не грають зовсім. В цьому випадку не можна з повною упевненістю говорити про причинно-наслідкові зв'язки. Можливо, що люди з певними когнітивними особливостями краще грають в відеоігри, тому їм це більше подобається, а люди, що не захоплюються іграми, не так гарні в певних характеристиках.

У інших дослідженнях усі випробовувані спочатку не вміють грати або грають однаково добре. У рамках дослідження вони проводяться через ряд дій, тобто грають в відеоігри, а потім порівнюються їх характеристики до і після. Встає питання про те, наскільки реально провести достатню кількість дослідів. У гравців із стажем 7-10 років кількість годин, проведених за грою, може бути величезною. У рамках дослідження, яке обмежене за часом, у кращому разі тестуються десятки годин гри. Можливо не отримати ніяких результатів або отримати ті, які не відіб'ють довгострокових ефектів відеоігри [16]-[17].

Існують суперечливі дані про те, наскільки просторові здібності розвиваються в грі. Останні нейропсихологічні дослідження, в яких випробовувані проходили спеціально організовані ігрові сесії, показують зміни в об'ємі сірої речовини в області гіпокампу, тобто в зонах, які пов'язані з орієнтацією в просторі. Це показано у гравців в тривимірні аркадні ігри і у деяких гравців в ігри жанру екшен. Також це відбувається не з усіма випробовуваними, а в залежності від стратегії, яку вони використовують.

Існують дві глобальні стратегії орієнтації в просторі, які виділяють в дослідженнях: орієнтація безпосередньо на просторові чинники і орієнтація на запам'ятовування, на автоматичні дії, пов'язані з орієнтацією в ігрових лабіринтах. Ефекти на мозок можуть кардинально розрізнятися залежно від переважаючої у людини стратегії. Проте зміни, які відбуваються в мозку у випробовуваних, необов'язково відбиваються на поведінці в реальному житті [16]-[17].

У іншому нейропсихологічному дослідженні не виявили ніяких відмінностей в мозковій активності людей, які мали великий досвід гри в відеоігри, і людей, які взагалі не мали досвіду. Але виявили відмінності на рівні точності виконання завдань. Гравці в відеоігри швидше і точніше справлялися із завданнями на розпізнавання візуальних і буквених стимулів. При цьому вони не демонстрували особливостей в роботі мозку, які б дозволяли говорити, що вони краще або гірше уміють вирішувати ці завдання на рівні нейронного

реагування. Знову йдеться про ігри жанру екшен, а це тільки один з жанрів, що ще більше ускладнює картину.

Про пам'ять і технології сказано багато. Пам'ять буває довготривала і короткочасна. Частіше говорять не про короткочасну, а про робочу пам'ять, яка потрібна людям, щоб здійснювати поточну діяльність і утримувати інформацію, якою вони оперують зараз. Довготривала пам'ять значно складніше для вивчення, тому що потрібні довгі дослідження. Але існує загальне припущення про те, що через сучасні технології здатність до довготривалого запам'ятовування інформації знижується. Усі стали покладатися на електронні пристрої, замість того щоб тримати в голові номери телефонів, свої зустрічі і різні справи [16]-[17].

Відеоігри не адресують довготривалу пам'ять безпосередньо. Вони адресують короткочасну робочу пам'ять. Існують дослідження, що показують збільшення об'єму робочої пам'яті у гравців в відеоігри. Це безпосередньо пов'язано з тим, що їм необхідно робити в грі, тому що гра досить складна діяльність, під час якої треба відстежувати різні показники і утримувати в голові багато інформації.

Гра спрямована на багатозадачність. Це ще один камінь спотикання сучасних інформаційних технологій і їх досліджень. Багатозадачність – загальний сучасний тренд. Багато досліджень показують, що для людей природно відстежувати декілька джерел інформації: читати прогноз погоди внизу екрану, слухати диктора, дивитися в телефон і одним оком ще і в книгу. Люди старшого покоління, навпаки, звикли робити по одній справі за раз. Поки що дослідження показують, що вони ефективніші [16]-[17].

Резолюція Американської психологічної асоціації показує, що при мультитаскінгу людина витрачає більше часу і ресурсів і сильніше навантажує нервову систему, тому що саме трудомістке в її діяльності – це перемикання між завданнями (task switching). Витрачається більше зусиль, щоб відволіктися від однієї справи і почати іншу. Тому що насправді люди не займаються двома справами одночасно – вони швидко перемикаються між ними.

Проти мультитаскінгу виступає багато дослідників. Існують дані, що при активному мультитаскінгу зменшуються об'єми деяких зон мозку і знижується глибина переробки інформації. Останнє абсолютно логічно, тому що кожному завданню відводиться менше уваги. Інші дослідження зв'язують мультитаскінг зі зниженням емпатії, рефлексії, з погіршенням продуктивності роботи людського мозку.

Але мультитаскінг є одним з важливих атрибутів нашої реальності, він потрібен не лише для ігор. Не робити декілька справ одночасно при сучасному темпі життя дуже складно. Гравці в відеоігри справляються з мультитаскінгом краще, і це може бути застосовне в їх повсякденній діяльності. Але в цілому мультитаскінг не дуже корисний для людини на тому рівні розвитку, на якому він знаходиться зараз. Можливо, через деякий час всі пристосуються [16]-[17].

Технології впливають на мозкову пластичність і дійсно можуть викликати зміни в структурах мозку. На цих ефектах ґрунтована ідея розвиваючих, серйозних ігор, спрямованих на розвиток мозку і когнітивних функцій. Відмінність між серйозними (прикладними) і розважальними іграми полягає в завданнях, які ставлять перед собою розробники. Можна сказати, що у гравців в серйозні ігри розвиваються навички, які вони можуть переносити на інші види діяльності. Серйозні ігри зараз є однією з індустрій, що інтенсивно розвиваються, і сфер діяльності, що активно вивчаються.

Проте є характерні проблеми. Серйозні ігри повинні залишатися досить цікавими і захоплюючими, щоб зберегти сильні сторони відеоігор. Вони повинні залучати людину до гри, а також мають бути позбавлені потенційних негативних ефектів, які ми бачимо у ігор жанрів екшен і шутер. Якщо є підозра, що ці ігри можуть підвищувати агресію, нехай це і не зовсім доведено, тоді не можна будувати відеоігри для тренування за цим принципом.

Одна загальна велика проблема для досліджень в області відеоігор – різні переваги публікацій (publication bias). Це переважна публікація певної групи статей. Статті, що показують наявність позитивного ефекту, завжди приймаються більш охоче, ніж статті з негативними результатами. Тому

дослідження, які б показували, що відеоігри не розвивають, мають менше шансів бути опублікованими.

Робіт, які порівнюють тренуючі і розважальні відеоігри, дуже небагато. В основному вони не виявляють особливої користі від серйозних ігор. Навчання за допомогою відеоігор, отримання знань – це інша сфера, яку треба вивчати окремо.

На відміну від акценту на негативних результатах відеоігор, таких як імпульсивне залучення до відеоігор, дослідження когнітивного тренування у людей з когнітивним дефіцитом показали, що характеристики елементів відеоігор допомагають тренувати когнітивні функції [16]-[17].

1.5 Зв'язок жанрів ігор із різними когнітивними функціями

На підставі чотирьох літературних джерел [18]-[21] виділено шість жанрів відеоігор найбільш впливаючих на когнітивний розвиток людини (див. табл. 1.2).

Таблиця 1.2 – Жанри відеоігор, що впливають на когнітивний розвиток

Жанри	Піджанри	Опис
1	2	3
Традиційні ігри (TG)		Настільні, карткові ігри у цифровому форматі.
Симулятори (SG)		Найточніше відтворення реальності. У грі відтворюються всі тонкощі та нюанси, можливі в аналогічній ситуації у житті.
Стратегії (SVG)	Стратегії в реальному часі (RTS)	Це стратегія, але те, що відбувається в реальному часі. Головна ознака: будівництво бази та битви у реальному часі.
	Покрокові стратегії (TBS)	Це стратегія, ігровий процес якої складається з послідовності фіксованих моментів часу, іменованих ходами, під час яких гравці здійснюють свої дії.

Продовження таблиці 1.2

1	2	3
Екшени (AVG)	Шутери від першої особи (FPS)	У таких іграх грають від першої особи. Там не лише стріляють, а й використовують техніку, пастки, ховаються, вступають у ближній бій.
	Шутери від третьої особи (TPG)	Ігри де камера знаходиться позаду героя і де ми бачимо його цілком.
Фентезі-ігри (FG)	Рольові ігри (RPG)	Ігри, у яких можна жити, вживатися у ролі героя, у яких ролі основних переваг виставляють атмосферу, сюжет, ігровий світ.
	Пригодницькі ігри (Adventure games)	Ігри в яких гравці вивчають та досліджують середовище гри в повільному темпі, зосереджуючись на вирішенні проблем.
Аркади (Arcade)		Ігри з ігровим процесом, типовим для ігор в ігрових автоматах, що вимагають від гравця насамперед швидкості, реакції та уваги.

Першим жанром є традиційні ігри, такі як головоломки, карткові та настільні ігри.

Другий жанр – це симулятори (наприклад, спортивні чи автомобільні ігри, Sims, що створюють міста чи спільноти).

Третій жанр – стратегії, які поділяються на стратегію в реальному часі та покрокову стратегію залежно від того, як відбувається уявний процес.

Четвертий жанр – це екшени, які діляться на шутери від першої особи та ігри від третьої особи залежно від погляду гравців у грі.

Наступний жанр – фентезі-ігри. Їх можна визначити як віртуальні ігри, в яких гравці досліджують ігрове середовище у відносно повільному темпі для вирішення проблем і які зосереджені на уяві, пропонуючи гравцям фантастичне середовище з певними правилами. Серед описаних піджанрів фентезі-ігор, рольові ігри є відправною точкою, у якій сформувалося поняття спільноти ігор. Масивні розраховані на багато користувачів мережеві RPG (MMORPG) – це

віртуальні ігри, в яких соціальні аспекти і аспекти участі підкреслюються за рахунок надання самої віртуальної гри як соціальної арени.

І останній жанр, позначений у літературі – аркади. Це жанр відеоігор, ігровий процес (геймплей) яких заснований на швидкій реакції гравця, мінімального ступеня свободи керованих персонажів та простому управлінні.

Хоча виявлено, що когнітивні функції тренуються за допомогою відеоігор протягом відносно короткого періоду, покращений тип когнітивних функцій залежить від жанрів відеоігор.

Першою когнітивною функцією є увага. Активні гравці в одну якусь гру краще утримували увагу, а гравці у випадковій, тобто різних відеоіграх показали покращення розділеної уваги. Порівняно з іншими жанрами, в які грають у повільному темпі, екшени тісно пов'язані з покращенням виборчої уваги, що належить до розподілу уваги на релевантну інформацію. Було виявлено, що гравці у шутери від першої особи ефективно розподіляють увагу за рахунок покращення процесу уваги "згори-вниз". Гравці екшенів та пригодницьких ігор показували невелике послаблення уваги, яке відноситься до нездатності виявити та обробити ціль, що згодом була представлена відразу після раніше обробленої мети. Але тренуючись грати в ігри жанру екшен, тільки не інших жанрів (наприклад, традиційних ігор та симуляторів), виправило цей момент. Крім того, виявлено, що поліпшення уваги, пов'язане з відеоіграми, супроводжують зміни в галузях мозку. Зміни в активації мозку показали, що гравці екшенів краще фільтрують інформацію та помічають важливу інформацію. Також ці ігри пов'язані з посиленням візуальної уваги, що відіграє важливу роль у ефективній обробці інформації [15].

Грунтуючись на взаємодії між зоровою увагою та робочою пам'яттю була визначена друга когнітивна функція – робоча пам'ять. Робоча пам'ять потрібна, щоб здійснювати поточну діяльність та утримувати інформацію, якою ми оперуємо зараз. Існують дослідження, що показують збільшення обсягу робочої пам'яті у гравців відеоігри. Це безпосередньо пов'язано з тим, що їм необхідно робити у грі, тому що гра є досить складною діяльністю, під час якої

потрібно відстежувати різні показники та утримувати в голові багато інформації. Коли було досліджено зв'язок між випадковими казуальними іграми з робочою пам'яттю та поліпшенням когнітивних функцій, поліпшення робочої пам'яті не було виявлено. Проте часта гра була пов'язана з поліпшенням можливостей робочої пам'яті. 20 годин навчання гри зі схованими об'єктами та матрицями пам'яті призвели до покращення просторової робочої пам'яті. 30 годин тренування в екшенах протягом 1 місяця порівняно з тренуванням в симуляторах призвели до покращення візуальної робочої пам'яті. Великий досвід гри в AVG був пов'язаний із найкращими візуальними можливостями робочої пам'яті. Гравці у FPS показали більш точну та швидку обробку релевантної інформації з більшою ємністю робочої пам'яті порівняно з негравцями. Гравці в екшени продемонстрували покращену ємність робочої пам'яті за рахунок ефективного розподілу уваги на важливу інформацію. Ці результати дозволяють припустити, що гра у відеоігри, особливо в AVG, може мати потенціал для покращення робочої пам'яті, що важливо для навчання навичкам та придбання знань. Однак, оскільки незрозуміло, чи є розбіжність між дослідженнями AVG результатом різної тривалості тренувань або різних аспектів робочої пам'яті, потрібні подальші дослідження [15].

Третя когнітивна функція – зорово-просторова функція, що стосується сприйняття, розпізнавання та маніпулювання візуальними стимулами (наприклад, зорово-моторною координацією, навичками навігації). Про поліпшення просторового сприйняття повідомлялося у гравців у тетріс, який можна класифікувати як один із традиційних ігор, а гра в TG (тобто логічні/головоломки) була пов'язана з об'ємом сірої речовини у ділянці гіпокампу та смугастого тіла. Було також виявлено, що AVG та SVG пов'язані з покращеною зорово-просторовою функцією. Десять годин тренування в AVG призвели до покращення навичок навігації за рахунок прийняття стратегії реагування, яка побічно вимірює/показує обсяг гіпокампу та смугастого тіла. Відповідно, гравці AVG, які навчалися грі в SuperMario протягом 2 місяців, показали покращення обробки просторової інформації та координації зорово-

моторної функції. Таким чином, було виявлено, що гра у відеоігри пов'язана з нейрональною пластичністю в областях мозку, що беруть участь у навігації та візуальній увазі. Хоча точна тривалість тренування для виявлення структурних змін у головному мозку не була визначена, передбачається, що ці відеоігри пов'язані з покращенням зорово-просторової функції [15].

Четверта когнітивна функція – це ймовірне навчання, яке відноситься до використання декларативної (експліцитної) пам'яті для вирішення невизначеності. П'ятдесят годин тренувань AVG новачків в відеоіграх підвищили ефективність використання не тільки візуальної, а й аудіодоступної інформації. Гравці AVG також показали більш високу активацію в ділянках мозку, залучених у візуальні образи, семантичну пам'ять та когнітивний контроль, порівняно з гравцями, які не грають в AVG. Більш висока активація в гіпокампі у гравців в AVG була пов'язана з більш вираженим використанням декларативного знання. Можна зробити висновок, що гра у відеоігри може покращити ймовірнісне навчання за рахунок ефективного використання доказів, представлених у середовищі відеоігри [15].

П'ята когнітивна функція – навички вирішення проблем. Навички вирішення проблем було покращено за допомогою ігор стратегій. Підлітки, які найчастіше грали у стратегічні відеоігри (наприклад, SVG, RPG та аркади) протягом 4 років середньої школи показали найкращі навички вирішення завдань. Більше того, участь у стратегічних відеоіграх покращила навички випускників (наприклад, навички вирішення проблем, спілкування), що свідчить про потенційну ефективність навчання на основі відеоігор. Однак було виявлено, що ігрові звички (наприклад, частота та час відеоігор, жанри ігор) не впливають на академічні навички старшокласників. Невідповідність між результатами, мабуть, полягала у різних шляхах виміру навичок вирішення проблем. Хоча було припущено потенційний позитивний вплив стратегій, необхідно провести подальші дослідження, що вивчають ступінь, в якому ці навички можуть бути покращені за допомогою відеоігор, та вивчити зміни у відповідних ділянках мозку [15].

Остання когнітивна функція – навчання, наприклад, другої мови. Не тільки серйозні ігри, але й розважальні дають гравцям можливість для мовної практики та засвоєння. Серед різних жанрів MMORPG сповнені можливостей взаємодії між гравцями цільовою мовою, що пропонується як ефективний метод для розмовної практики і, як повідомляється, полегшує навчання мови. Було виявлено, що гравці, які беруть участь у цих відеоіграх, демонструють посилену функціональну зв'язність у ділянках мозку, що беруть участь у обробці промови. Більше того, зсув уваги до інформації, що стосується завдання, було визначено як можливий механізм полегшеного навчання мови в MMORPG порівняно з іншими іграми.

Взяті разом, шість когнітивних функцій були ідентифіковані як позитивно пов'язані з грою в відеоігри, незважаючи на деякі розбіжності між результатами. Різні жанри ігор пов'язані з різними аспектами когнітивної функції [15].

1.6 Модуючі фактори

Було продемонстровано позитивний зв'язок між грою в відеоігри та когнітивними функціями. Проте є індивідуальні розбіжності у ступені поліпшення когнітивних здібностей гравців. Це пов'язано з тим, що деякі фактори впливають на пластичність та індивідуальні реакції на тренування у відеоігри. Існує п'ять факторів, які визначені для модуляції зв'язку між грою в відеоігри та когнітивним покращенням.

Перший модулюючий фактор – досвід гри у відеоігри. Гравці з великим досвідом гри віддавали пріоритет навичкам для вироблення стратегії під час гри порівняно з новачками, які віддавали пріоритет навичкам дій. Тобто, досвід вплинув на активацію різних когнітивних процесів під час гри. Більш того, було виявлено, що ігровий досвід впливає на переваги, що пов'язані з цією грою [15].

Другим певним модулюючим фактором індивідуальних відмінностей є вік гравців. Діти приділяють іграм багато часу. Після досягнення 13-14 років час гри у відеоігри з віком скорочувався. Вікова різниця в ігровому часі передбачає, що відеоігри потенційно мають більший вплив на покращення когнітивних функцій у молодих людей, ніж у людей похилого віку. Вік тісно пов'язаний з когнітивними функціями та виконанням завдання, оскільки функціональні зв'язки мозку розвиваються з віком. Хоча значну нейропластичність було виявлено у дітей молодшого віку, також було виявлено вікове зниження когнітивного контролю. Виконання завдань у дітей молодшого віку з відносно повільною та менш точною увагою стало краще [15]. Дослідження доводять, що 38% дорослих старше 50 років грають у відеоігри. Будь то онлайн через ноутбук або настільний комп'ютер, ігрову консоль або застосунок для смартфона, віртуальні ігри корисні для повторення завдань, орієнтованих на когнітивні здібності [4].

Третім фактором є базова когнітивна функція (наприклад, вміння розмірковувати, навичка уваги). Ця функція впливає на вибір відеоігор, оскільки когнітивні здібності були кращими у гравців, які обрали аркадну гру,

ніж у людей, які ніколи не грали у ігри. Це також впливає на ступінь, в якому когнітивні функції будуть поліпшені за допомогою гри. Було виявлено, що діти з великим дефіцитом уваги покращують її завдяки комп'ютеризованому навчанню та демонструють стійкий ефект протягом 9 місяців. Також було виявлено, що гравці з нижчими базовими здібностями до міркування отримують більше когнітивних переваг, таких як кращий поділ уваги та швидке сприйняття стимулів. Цілком ймовірно, що модулююча роль базової когнітивної функції залежить від аспекту когнітивної функції, що тренується у грі. Хоча для вивчення цієї ідеї мають бути проведені подальші дослідження. Передбачається, що вихідна когнітивна функція модулює когнітивну користь від гри, впливаючи на вибір жанру та ступінь когнітивного покращення [15].

Когнітивна функція, яка була визначена як один із модулюючих факторів, пов'язана зі статтю в тому сенсі, що чоловіки краще пригнічували відволікаючі фактори та утримували увагу. Тобто, четвертим певним фактором є стать, яка впливає на ігрові звички, які, як повідомляється, пов'язані зі ступенем покращення та типами покращених когнітивних функцій. Хоча і жінки, і чоловіки в екшенах отримали однакову перевагу щодо уваги, стать впливає на час і стиль гри. Збільшення часу гри у чоловіків було пов'язане із жанром ігор (наприклад, за участю кількох гравців). Більше того, чоловіки вибрали комп'ютер як ігрову платформу порівняно з жінками. Хоча різниця в поліпшенні когнітивних здібностей між ігровими платформами (тобто мобільними та консольними тощо) не була значною, було виявлено, що вибір ігрової платформи пов'язаний з мотивацією (наприклад, соціальна взаємодія) для участі у відеоіграх. У той час як чоловіки вважають за краще грати зосередившись на змаганні (наприклад, AVG або SG), жінки люблять грати у TG та FG. Тобто, стать, яка пов'язана зі звичками залучення до ігор, побічно модулює зв'язок між грою та покращенням когнітивних функцій [15].

На основі гендерних відмінностей у виборі ігрової платформи, останній фактор, який модулює зв'язок між грою та когнітивними функціями – мотивація. Гравці з більш високим рівнем мотивації брали участь у тренуваннях

добровільно, краще виконували тренування та демонстрували вищий рівень робочої пам'яті, ніж люди з нижчим рівнем мотивації. Гравці, які були мотивовані до спілкування в іграх завдяки досвіду отримання соціальних винагород (наприклад, позитивних емоцій в ігровому середовищі), також демонстрували увагу до навчання мови. Більше того, наявність мотиваційного фактора під час гри впливає на функціональні зміни мозку, пов'язані із тренуванням. Гравці, які зазнали більше задоволення та відносно менше розчарувань під час гри були більш мотивовані грати далі та показали більше функціональних змін у відповідних ділянках мозку [15].

Оскільки п'ять факторів, визначених як такі, що впливають на індивідуальні відмінності в асоціації між грою і когнітивним покращенням, взаємодіють один з одним у модуляції асоціації, важко зробити висновок, який фактор має більший вплив на індивідуальні відмінності у зв'язку між грою та когнітивними поліпшеннями. Більше того, є інші фактори, які не розглядаються у цьому розділі. Наприклад, особистості гравців впливають на мотивацію гри в відеоігри та на вираження мотивації під час гри. Риси особистості також можуть впливати на ефект навчання, тому що гравці-інтроверти можуть отримати більше переваг від вивчення мови, граючи в ігри, де вони практикуються у спілкуванні більш вільно, ніж традиційне навчання. Більше того, гравці показали різницю у результатах навчання, залежно від свого стилю навчання.

Відеоігри, метою яких є розвага гравців, виявилися позитивно пов'язані з когнітивними функціями (наприклад, увагою, навичками вирішення проблем), незважаючи на деяку розбіжність між дослідженнями. Більш того, оскільки було виявлено кілька факторів (наприклад, вік, стать) для модуляції когнітивного покращення, було виявлено індивідуальну різницю у зв'язку між грою у відеоігри та когнітивною функцією [15].

1.7 Огляд існуючих аналогів

Аналоги були підібрані по вікових категоріях, по процесу прийому і зміни інформації з довкілля, а також асиміляції, як люди співвідносять нову інформацію з раніше існуючою інформацією. Він торкається усіх гравців і фокусується на прогресі розвитку, а не на вивченні різних форм поведінки. Відзначається прогресія когнітивного розвитку, починаючи з дитинства і упродовж усього дитинства, отрочтва. У міру дорослішання людини зростає і його здатність брати участь у вищих розумових процесах, таких як рішення проблем, емоційна регуляція, навчання і запам'ятовування. Цей термін описує розвиток мозку стосовно пізнання.

1.7.1 Вебзастосунок «CogniFit»

Одним з аналогів розроблюваної програми є вебзастосунок "CogniFit". Інтерфейс зображено на рис. 1.1. У ньому досить велика різноманітність ігор, що розвивають когнітивні функції, такі як пам'ять, сприйняття, міркування, координація і увага [22].



Рисунок 1.1 – Користувачський інтерфейс вебзастосунка «CogniFit» [22]

1.7.2 Програмний продукт «Sea Hero Quest»

З усіх когнітивних навичок пам'ять – це навичка, на яку найбільше впливає старіння. Гра в ігри, в яких основна увага приділяється завданням з пам'яттю, що знижує міру втрати пам'яті. Літнім людям з деменцією особливо корисні ігри, спрямовані на пам'ять.

Аналогом приведено гру для мобільних пристроїв "Sea Hero Quest", в якій необхідно запам'ятати маршрут по карті, щоб досягти точки призначення або знайти монстра. Інтерфейс зображено на рис. 1.2. Ця гра була створена в співпраці з британськими ученими з метою вивчення одного з різновидів синдрому Альцгеймера – деменцією. За статистикою близько сорока п'яти мільйонів чоловік по всьому світу страждає деменцією. Це захворювання проявляється у втраті пам'яті, погіршеній здатності адаптуватися в просторі і неможливості виконання логічних завдань. Незважаючи на велику кількість хворих цією хворобою, детальних даних критично мало і Sea Hero Quest може стати прекрасним джерелом інформації [23].

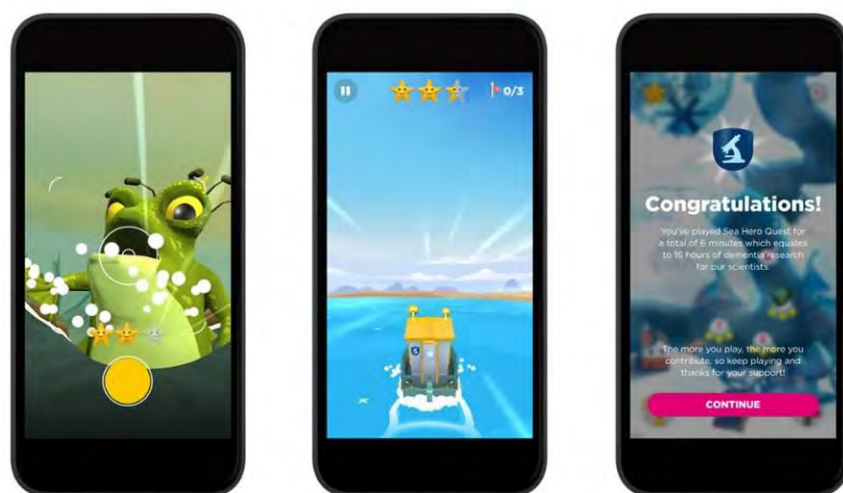


Рисунок 1.2 – Інтерфейс мобільного застосунку «Sea Hero Quest» [23]

1.7.3 Програмний продукт «Geometry Dash»

Наступним аналогом є аркадна гра "Geometry Dash" [24], що націлена на аудиторію як підлітків, так і літніх людей. Інтерфейс зображено на рис. 1.3. Це гра безпосередньо розвиває швидкість реакції. Гемплей гри складає спроби дістатися до кінця рівня за допомогою квадратики і ряду інших засобів, минуючи перешкоди.



Рисунок 1.3 – Користувацький інтерфейс «Geometry Dash» [24]

1.7.4 Порівняння існуючих та розроблюваного програмних продуктів

В таблиці 1.3 проведено порівняння вже існуючих програмних продуктів та програми «Cross Way Driving», яка буде розроблена в ході виконання роботи.

Таблиця 1.3 — Порівняння програмних продуктів

Властивості	Назва застосунку			
	вебзастосунок «CogniFit»	«Sea Hero Quest»	«Geometry Dash»	«Cross Way Driving»
1	2	3	4	5
Інтерфейс	7	9	10	10
Зручність	8	8	9	10

Продовження таблиці 1.3

1	2	3	4	5
Дизайн	10	9	8	10
Інтуїтивна зрозумілість	9	9	10	9
Складність	8	8	10	10
Стабільність	6	7	8	9

Було наведене популярні на сьогоднішній день відеоігри, які дозволяють розвивати когнітивні функції людини.

Оцінки встановлені згідно коментарям та оцінкам користувачів ігор.

Вебзастосунок «CogniFit» має непоганий дизайн та зручність використання, але через інтерфейс не є інтуїтивно зрозумілим, а стабільність не є кращою. Самі ігри не дуже складні, але виконують потрібну функцію.

Мобільний застосунок «Sea Hero Quest» має непогану стабільність та інтерфейс з дизайном, завдяки чому досягається зручність та інтуїтивна зрозумілість, але складність ігри не є кращим.

«Geometry Dash» один з кращих аналогів, нарікань майже немає, є невеликі похибки в дизайні та гіршу стабільність. Має непогану складність гри, з часом вона підвищується.

Мобільна гра «Cross Way Driving» відрізняється тільки невеликою похибкою в стабільності роботи, яка теж присутня в інтуїтивної зрозумілості, але все інше привабливе.

1.8 Аналіз цільової аудиторії

Ця гра призначена для дітей дошкільного віку. Вона схожа на гру дітей у іграшкові машинки з примітивними функціями: прискорення, уповільнення та поворот машинок у побудованому, припустимо з конструктора, місті. Коли діти грають вони не дотримуються правил дорожнього руху, але можуть спостерігати за машинками і по миготінню поворотника зрозуміти куди поверне машинка і чи потрібно її прискорювати, щоб не сталося зіткнення. Гра націлена

на розвиток когнітивних функцій: уваги, логічного мислення та реакції. В більш старших людей відповідно є інші можливості для розвитку цих когнітивних функцій і відповідно рівень гри вже недостатній для віку більше ніж 7 років.

1.9 Постановка задач роботи

Як показали проведені дослідження, відеоігри допомагають тренувати когнітивні функції. Розглянуті жанри ігор пов'язані з різними аспектами когнітивних функцій.

В результаті проведеного аналізу було вирішено розробляти мобільну гру для Android. За жанром розроблювана гра буде аркадою для розвитку таких когнітивних функцій, як швидкість реакції, увага, логічне мислення.

Мета роботи – дослідити та розробити 3D гру для Android, що впливає на розвиток когнітивних функцій.

Задачі роботи:

- виконати аналіз вимог до програмного забезпечення;
- спроектувати структуру гри та обрати інструментарій розробки;
- виконати програмну реалізацію;
- провести тестування гри та виконати її оптимізацію;
- розробити програмну документацію.

1.10 Висновок за першим розділом

У розділі було розглянуто актуальність проблеми, поняття та основні функції головного мозку, як він працює. Розглянуто поняття когнітивного розвитку та його областей. Також було розглянуто дослідження відеоігор з точки зору когнітивного розвитку, вивчення впливу ігор на когнітивний розвиток людини. Крім того, показано зв'язок жанрів ігор із різними когнітивними функціями та описано модулюючі фактори, що визначені для модуляції зв'язку між грою в відеоігри та когнітивним покращенням. Таким

чином, це дослідження було спрямоване на те, щоб мати збалансованіший погляд на відеоігри, розглядаючи жанри розважальних відеоігор і зв'язок відеоігор з когнітивними функціями і модулюючими чинниками. Було проведено огляд популярних ігор для розвитку когнітивних функцій та порівняно у вигляді таблиці. Було обрано операційну систему Android для якої буде написана гра. Мобільна платформа є найпопулярнішою, тому що саме на цій платформі можна грати всюди, починаючи від домашнього дивана – закінчуючи пробіжкою лісом. У той час як персональні комп'ютери та консолі не можуть похвалитися такою (як це не очікувано) мобільністю та перебувають лише на одному місці. Також ігри на цій платформі не є чимось напруженим. Найчастіше це просто аркади та клікери, які не змушують сильно про щось замислюватися або концентруватися на чомусь. У той час як більшість ігор на персональному комп'ютері та консолях можна назвати певною мірою важкими, а іноді навіть випробуваннями. Гра орієнтована на дітей дошкільного віку, тому що для розвитку когнітивних функцій рівень гри вже недостатній для віку більше ніж 7 років.

2 ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ЙОГО ОБҐРУНТУВАННЯ

2.1 Вибір ігрового рушія

Ігрові рушії – це платформи програмної розробки, що включають всю необхідну функціональність для створення відеоігор, від розробки сценарію до застосування штучного інтелекту, фізики та рендерингу [25].

Unity і Unreal Engine 4 (UE) – це одні з найпопулярніших ігрових движків, доступних на сьогоднішній день. У кожного з двигунів є свої сильні сторони для різних завдань. Вибір движка залежить від мети, бачення та завдань розробника.

Unreal набагато частіше використовується в іграх для ПК, у той час, як Unity домінує на ринку мобільних ігор.

З роками і Unity, і Unreal Engine усували свої слабкі місця, тож вибір одного з них над іншим – непросте завдання [26].

2.1.1 Опис Unity3D

Unity – дуже універсальний рушії, який стає основним продуктом для мобільних розробників. Це механізм розробки, вперше випущений 2005 року. З того часу він став найпопулярнішою платформою для розробки 3D та 2D у світі.

Ось лише деякі статистичні дані:

- 60% AR/VR-контенту та 50% мобільних ігор створюються за допомогою Unity3D;
- більше 24 мільярдів установок рушія за рік;
- підтримує 28 платформ (від iOS та Android до Oculus та Windows Mixed Reality та всіх проміжних).

За останні кілька років рушій настільки просунувся вперед, що тепер він є багатогранною платформою розробки, яку можна використовувати для будь-якого складного XR-рішення [27].

Переваги Unity:

- Unity безкоштовна. Існує 2 безкоштовні версії: Unity Personal та Unity Student. Unity Personal призначена для приватних осіб, любителів та невеликих організацій з доходом менше 100 000 доларів за останні 12 місяців. Unity Student – для учнів від 16 років, які навчаються в акредитованому навчальному закладі [28];

- простота використання. Рушій використовує C#, який вважається простим, має дуже інтуїтивно зрозумілий інтерфейс та просту архітектуру (об'єкти та компоненти). Крім того, він має інструмент візуального створення сценаріїв Bolt, що робить Unity ще простіше у використанні. По суті це дозволяє користувачам Unity максимально уникати написання коду;

- складання для різних платформ. Безкоштовна версія Unity дозволяє збирати програми для macOS, ПК, інтернету (з використанням WebGL), Linux, iOS, Apple tvOS, Android, Samsung TV, Tizen, Windows Store та багатьох інших платформ – з одного й того самого коду та файлів;

- швидкість. Unity відома своїми швидкими циклами ітерацій, частково через свою здатність мінімізувати використання коду;

- спільнота. Понад 200 тисяч учасників на офіційному сабреддіті. Що дає велику базу знань [29]-[30].

Недоліки Unity:

- гібрид коду з візуальним редактором, незважаючи на ефективність в якості компонентної системи. Unity – не має строгих типів, і це може створити не малі проблеми. Складні сцени можуть не дозволяти бачити всю сформовану картину, і розсіюють увагу програміста. Unity має функцію пошуку, яка дозволяє знаходити сценарії, але вона могла бути набагато надійнішою – періодично трапляються казуси, у яких під час пошуку пов'язаних скриптів програміст змушений переглядати кожен елемент сцени. Хоч і трапляється це

не часто, але це дуже не приємна та кропітка робота, яка займає багато енергії [29]-[31];

– наступним недоліком можна визначити те, що в Unity не можна додавати посилання на зовнішню бібліотеку. Всі зовнішні бібліотеки потрібно копіювати вручну, і додавати до проекту, хоча можна було б просто залишати посилання на джерело із загального доступу. Єдина папка з бібліотеками дозволила б команді розробників використовувати функціонал одночасно різними проектами. Цей дискомфорт можна обійти, використовуючи правильно системи контролю версій, але рішення з коробки відсутнє [29]-[31];

– останній недолік полягає у використанні шаблонів екземплярів (prefabs). Шаблони екземплярів пропонують підхід, у якому досягається гнучкість візуального створення інтерактивних об'єктів. Такий, безумовно, потужний концепт не має аналогів і виходить від компонентної системи Unit, проте виправлення таких шаблонів час від часу виявляється неймовірно складним [29]-[31].

2.1.2 Опис Unreal Engine 4

Unreal Engine має тривалу історію ігрового движка, починаючи з 1998 року. Це ігровий рушій, який справив фурор в індустрії відеоігор як один з перших, хто підтримує DirectX Raytracing (DXR), дозволяючи використовувати тільки трасування променів на тих проходах, які її потребують, і розтеризацію на тих, які її не потребують. Це гарантувало дотримання як якості рендерингу, і швидкості рендерингу. З тих пір UE 4 робить величезні кроки для покращення швидкості та якості рендерингу з кожним наступним оновленням.

Нижче наведено ключові статистичні дані для Unreal Engine 4:

- понад 7 мільйонів користувачів з дизайнерської та корпоративної спільноти;
- більше 15 підтримуваних платформ.

У наші дні незалежні розробники та великі студії можуть використовувати Unreal Engine для своїх різних проєктів, включаючи ігри, освітні системи та бізнес-рішення.

UE 4 можна використовувати для розробки VR/AR, 2D-ігор, 3D та мобільної розробки [27].

Переваги Unreal Engine 4:

- візуальне програмування. UE 4 має вкрай продуктивну і відкалібровану систему візуального скриптування, під назвою Blueprints, яка дозволяє досягти майже тих самих результатів, як і за допомогою C++. Завдяки такому програмуванню можна заощадити величезну кількість часу, при тому самому створенні персонажа та його налаштування;

- ассети UE орієнтовані на готові ігрові продукти. Багато хто з них раніше використовувалися в AAA-іграх. Це пояснює їхню високу візуальну якість;

- мультизадачність та універсальність. Цей рушій, дійсно, настільки універсальний, що на ньому можна спокійно робити спецефекти до фільмів. А якщо брати архітектурну візуалізацію, існує окремий проєкт Twinmotion [29]-[30].

Недоліки Unreal Engine 4:

- найголовніший мінус – завищена ціна у магазині на контент. Дуже часто ціна на потрібні речі виявляється близько 100 \$. Та й до того ж бувають проблеми із сумісністю;

- графіка не найкраща. UE генерує все ж таки трохи пластилінову графіку, особливо рослинності, води, шкіри. Саме тому ігри на UE краще виходять із мультяшною графікою. Також дуже мало універсального контенту, в порівнянні з тим же Unity;

- вимоги до системи. Справа в тому, що UE сильно навантажує ПК. Особливо, якщо проєкт має не оптимізовану графіку та код. Постійно щось кешує, компілює, перераховує. Краще ставити одразу сервер;

– складний процес роботи з ландшафтами, особливо текстуровання. Куди грамотніше і простіше це реалізовано у тому ж Unity [29]-[30].

2.1.3 Порівняння ігрових рушіїв

Нижче наведено порівняльну таблицю таких ігрових рушіїв, як Unity та Unreal Engine (табл. 2.1).

Таблиця 2.1 – Порівняння ігрових рушіїв Unity та Unreal Engine

Параметри порівняння	Ігровий рушій	
	Unity	Unreal Engine
Платформи, що підтримуються	Великий вибір	Великий вибір
Графіка	Висока. Але більше роботи із графікою.	Відмінна
Відкритий вихідний код	Тільки для тих, хто купив платну версію.	Для всіх абсолютно безкоштовний та доступний на гітхабі.
Кількість навчальних матеріалів	Багато	Мало
Кількість ассетів (моделі, анімації, шейдери тощо)	Величезний вибір платних та безкоштовних ассетів	Менший вибір платних та безкоштовних ассетів
Для кого рушій	Для невеликої команди або одного розробника	Для великої команди

Платформи, що підтримуються. Тут обидва двигуни показують себе добре, і проєкт можна зібрати під різні операційні системи і різні пристрої.

Графіка. Графіка не найкраща, як то кажуть. UE генерує все ж таки трохи пластилінову графіку, особливо в частині рослинності, води, шкіри. Саме тому ігри на UE краще виходять із мультяшною графікою.

Відкритий вихідний код. Тільки для тих, хто платить 150 \$ на місяць і має річний оборот мінімум 200 000 \$. У UE для всіх абсолютно безкоштовний та доступний на гітхабі.

Кількість навчальних матеріалів. У Unity їх більше, ніж у UE. Пов'язано це з тим, що Unity у вільному доступі вже дуже багато років (з 2005-го року), тоді як Unreal став доступним усім охочим лише у 2015 році [29]-[30].

Кількість ассетів (моделі, анімації, шейдери тощо). У Unity їх більше, ніж у UE. Ассети UE, як правило, орієнтовані на готові ігрові продукти. Це пояснює їхню високу візуальну якість. Серед цих активів є замки, науково-фантастичні технології та монстри. Ассети для Unity теж непогані. У магазині Unity є можливість вибирати з більшої кількості ресурсів, простіше знайти більш підходящі моделі та швидко створити прототип гри без необхідності наймати фахівців з 3D-моделювання та створювати віртуальні об'єкти з нуля. Крім того, у них доступні ціни на платний контент [27].

Для кого рушій. Unity призначено для використання невеликою командою або одним розробником. UE був спеціально створений для великої команди – студій, які випускають такі ігри, як Gears 5 та інші класичні блокбастери.

У плані роботи Unity зроблений простим, а UE – ефективним. Це простежується у роботі з файлами, як пишеться ігрова логіка, навіть у тому, як виглядає стартовий проєкт.

В результаті порівняння було обрано Unity, як ігровий рушій для майбутньої гри.

2.2 Вибір мови програмування

На движку Unity можна створювати ігри, використовуючи різні мови програмування. У кожній мові програмування власні особливості та призначення, тому назвати якийсь із них єдиною необхідним не можна. Офіційно на вибір пропонується три варіанти: Boo, C# та JavaScript (UnityScript). Нерідко під час розробки використовують відразу кілька мов. І пишуть одні файли сценарію однією мовою, а інші – іншою, фактично змішавши мови. Unity не забороняє так діяти. Проте це погано, тому що подібна практика зазвичай

призводить до плутанини, а також до конфліктів при компіляції. Це все одно, що намагатися розрахувати відстань у милях та кілометрах одночасно. Рекомендований підхід полягає в тому, щоб вибрати одну мову та використовувати її всюди в проєкті як головну мову. Це спростить роботу, але це також означає, що доведеться вибрати одну мову, а від інших відмовитись. Найчастіше використовуються C# або UnityScript [32].

2.2.1 Мова програмування C#

C# – це мова програмування з високою продуктивністю, розроблена Microsoft та працююча на платформі .NET Framework. C# використовується для розробки вебзастосунків, настільних застосунків, мобільних застосунків, ігор та багато іншого. Крім того, ця мова програмування забезпечує високу апаратну сумісність. Якщо знати цю мову програмування, можна використовувати потенціал Unity на 100%. В основному, всі бібліотеки Unity написані саме на C#. Unity використовує open-source проєкт Mono, який є одним із численних реалізацій платформи Microsoft .NET. C# – досить проста в освоєнні мова, до того ж не потрібно знати її досконально. Для роботи з Unity достатньо базових навичок. Його синтаксис схожий з C++, Java, тому якщо розробник володіє однією з цих мов, то освоїти C Sharp буде ще простіше [33].

2.2.2 Мова програмування UnityScript

UnityScript – ще один оптимальний варіант, якщо розробник планує працювати з ігровим рушієм Unity. Він має досить простий синтаксис, завдяки чому освоїти мову буде досить легко. Мова дуже схожа на JavaScript, але є певні відмінності, наприклад, наявність класів та відсутність функції перепризначення змінних, відсутня можливість перепризначення змінних, не можна використовувати необов'язкові точки з комою. На жаль, підтримку

UnityScript постійно видаляють із ігрового движка. Розробники отримують попередження про необхідність перевести весь код скриптів C# [33].

2.2.3 Мова програмування Boo

Boo – мова, від якої відмовилися розробники Unity, що має короткий та зручний синтаксис, який підтримується .NET та Mono. Синтаксис Boo – щось середнє між IronPython та CPython, через налаштування компілятора його можна зробити динамічно-типізованим, хоча за умовчанням він таким не є. Тобто, за структурою вона схожа на Python і проста в освоєнні. Мова проста і зручна, синтаксис не перевантажений складними пропозиціями.

У Unity вона підтримувалась доти, доки над платформою працював його автор. Від мови відмовилися з банальної причини – через низьку популярність. Boo не привносив нових можливостей у Unity, не приваблював армію розробників, тому його підтримка була жестом доброї волі [33].

2.2.4 Мови програмування IronPython и IronRuby

Для любителів Python та Ruby все ж таки є можливість попрацювати з Unity. Обидві мови – результат реалізації Python та Ruby відповідно на платформі .NET. Щоб використовувати їх у Unity, доведеться трохи повозитися з підключенням потрібних бібліотек, їх можна знайти на GitHub. Усі матеріали у вільному доступі. Після підключення обох мов можна буде викликати скрипти, написані ними, прямо з коду C#. Бонусом буде можливість виклику .NET бібліотек із Python.

Рішення – IronPython, набір бібліотек, розташованих у GitHub. Дозволяє запускати "змійні" скрипти прямо з C#-коду. Корисне розширення – можливість викликати .NET-бібліотеки з Python.

Аналогічну функціональність мають бібліотеки під загальною назвою IronRuby. Синтаксис схожий, тому доведеться витратити час лише на

підключення вихідних джерел, зате у відповідь Unity підтримуватиме ще одну мову – Ruby [33].

2.2.5 Мова програмування Lua

Lua використовується переважно в іграх, так що в Unity має бути представлений. За це відповідає MoonSharp – інтерпретатор для Mono, платформи .NET та Unity. MoonSharp не замінює C#, а дає розробнику можливість підключити сторонній код Lua. Величезна кількість модів, рівнів та розширень у багатьох іграх написано саме на Lua. MoonSharp не слід розглядати як заміну C# [33].

2.2.6 Порівняння мов програмування

Вибираючи між мовами програмування можна відразу відкинути мову Boo, тому що вона не розвивається і не підтримується розробниками. IronPython та IronRuby теж можна відкинути. Вони використовуються розробниками, які віддають перевагу Python і Ruby. А мова програмування Lua вважається необов'язковим доповненням. За допомогою неї можна привнести до своїх ігор деяку модальність.

Для порівняння залишаються лише UnityScript та C#, що подано в табл. 2.2.

Таблиця 2.2 – Порівняння мов програмування

Властивості	Мови програмування	
	C#	UnityScript
Швидкість	Висока	Висока
Читабельність	Відмінна	Добра
Сумісність з VS або MonoDevelop	Так	Немає
Кількість бібліотек у вільному доступі	Багато	Багато

Продовження таблиці 2.2

1	2	3
Підтримка спільноти	Так	Немає
Строго типізований	Так	Немає

Швидкість. У Unity кожного разу, коли розробник зберігає файл з вихідним кодом на C# або JavaScript, Unity імпортує його, перетворює на код однією і тією ж узагальненою проміжною мовою Common Intermediate Language (CIL) і потім компілює код на CIL в код машинною мовою. Тобто незалежно від вибраної мови прототипи ігор в Unity будуть виконуватися з однаковою швидкістю [34].

Читабельність. З C# код більш читальний і зручний у супроводі.

Сумісність із Visual Studio (VS) або MonoDevelop. Редактори (VS або MonoDevelop) можуть надати найкращі пропозиції з C#. Інші функції VS, такі як створення графіків з коду, пошук посилань тощо, не працюватимуть з UnityScript.

Кількість бібліотек у вільному доступі. Існує безліч бібліотек, доступних як на C# так і на UnityScript [35].

Розробники двигуна офіційно заявили, що насамперед підтримують C#, а з літа 2017 року посилено викоринюють UnityScript з платформи. Авторам пакетів в Asset Store, що містять UnityScript, було направлено прохання про зміну коду на C#. Вже більшість прикладів, зразків та плагінів використовують C#.

C# строго типізований, чого не можна сказати про JavaScript. JavaScript використовує слабку типізацію, тобто створивши змінну (або контейнер), можливо зберегти в ній все що завгодно: число, слово, цілий рядок і навіть головного персонажа вашої гри. Так як ця змінна в JavaScript не має типу, Unity ніколи не зможе точно знати, що в ній дійсно зберігається, а крім того, вміст цієї змінної може змінитися в будь-який момент. Така гнучкість JavaScript робить написання сценаріїв більш стомлюючим заняттям і не дозволяє

програмістам користуватися деякими потужними та цікавими можливостями сучасних мов. Крім того, в JavaScript синтаксис вибачає помилки [34].

Розглянувши усі переваги та недоліки мов програмування було обрано C#.

2.3 Вибір середовища розробки

При виборі редактора коду слід враховувати плюси та мінуси. А також, чому віддає перевагу розробник: функціональності або продуктивності редактора. Обидва IDE прекрасно підходять для створення чудових ігор на Unity.

2.3.1 Опис MonoDevelop

Monodevelop – це IDE з відкритим вихідним кодом, яка доступна для широкої публіки безкоштовно для розробки вебзастосунків та програм для настільних комп'ютерів. Він працює в системах MacOS, Windows та Linux і дозволяє розробнику в основному зосередитися на моно-застосунках або застосунках .NET. Monodevelop підтримує різні основні мови, такі як C, C#, Java, F#, Javascript тощо. Він забезпечує дуже простий, швидкий і легкий спосіб запуску застосунків з використанням мінімальних ресурсів. А також не дозволяє використовувати багато функцій форматування, оскільки орієнтований на потреби бізнесу [36]. MonoDevelop дозволяє швидко розробляти настільні та вебзастосунки. IDE має здатність переносити програми .Net, які написані в Visual Studio на інші платформи, такі як Linux і Mac OS, у зв'язку з чим підтримує єдину базу для кодів платформ.

Переваги Monodevelop:

- мультиплатформність;
- широкий спектр кастомізації продукту;
- наявність відладчика та інших корисних функцій;

- повна підтримка unity;
- повністю безкоштовний продукт та вільний функціонал.

Недоліки Monodevelop:

- платформа має сирий функціонал, тобто платформа не може підтримувати різні проєкти та не супроводжує створенню комплексних застосунків;
- існують безглузді помилки в отладці [37].

2.3.2 Опис Visual Studio

Visual Studio (VS) – це багатофункціональне IDE з відкритим вихідним кодом від Microsoft, яке доступне широкій публіці для налагодження та складання коду, а також тестування та розгортання програм на різних платформах. Воно дозволяє розробнику розробляти мобільні застосунки, вебсайти, вебсервіси, забезпечуючи підтримку більш ніж 30 мов, таких як C, Visual Basic, JavaScript, C++, Visual Basic, Python і т. д. Крім того, воно надає користувачеві різні плагіни коду, рефакторинг та завершення коду. Інтегрується з багатьма інструментами для полегшення роботи розробника, єдиним недоліком якого є споживані ресурси і низька швидкість виконання коду [36]. VS визнана найкращою IDE для C#. Справа в тому, що обидва продукти належать корпорації Microsoft. Тому вони ідеально підходять для роботи один з одним.

Переваги VS:

- середовище містить багато інструментів, які дуже добре працюють на C#;
- наявність безкоштовної версії – Community Edition, яка містить все, що потрібно для незалежного розробника;
- найефективніше програмне забезпечення для розробки на будь-якій платформі, включаючи .Net і C#;
- можливість зберігання даних у хмарі;

– цілком підтримує популярну платформу для створення комп'ютерних ігор Unity 3D.

Недоліки VS:

- вибагливість до ресурсів;
- після переходу на платну версію, можуть злетіти налаштування та корпоративний сервер;
- складність при самостійному освоєнні [37].

2.3.3 Порівняння середовищ розробки

Нижче наведено порівняльну таблицю, що показує пряме порівняння між Monodevelop і Visual Studio (табл. 2.3).

Таблиця 2.3 – Порівняльна таблиця MonoDevelop та Visual Studio

Властивості	Середовища розробки	
	Monodevelop	Visual Studio
Продуктивність	Добре	Є проблеми
Стабільність	Найменш стабільне	Стабільніше
Плагіни	Немає	Так
Стилізація коду	Немає	Так
Підтримка спільноти	Має меншу підтримку спільноти	Має велику підтримку спільноти

Продуктивність. Monodevelop порівняно швидше, легше, тому споживає набагато менше ресурсів. Крім того, Visual Studio споживає багато ресурсів під час запуску будь-якої програми.

Стабільність. Monodevelop менш стабільне середовище розробки у порівнянні з Visual Studio. Це добре під час роботи з невеликими проєктами. Visual Studio може працювати з усіма типами проєктів, як маленькими, і великими.

Плагіни. У Monodevelop IDE програмісту не надаються жодні плагіни. Але Visual Studio надає можливість встановлювати різні платні та безкоштовні плагіни для розширення функціональності.

Стилізація коду. Якщо говорити про стилі, Monodevelop не надає розробнику можливості стилізувати код різними способами. Він показує лише простий спосіб кодування та нічого більше. Visual Studio надає розробнику можливість стилізувати код у відповідності до улюбленого зовнішнього вигляду.

Підтримка спільноти. Люди в усьому світі більше віддають перевагу Visual Studio як IDE для розробки коду. Таким чином, він має велику підтримку спільноти в порівнянні з Monodevelop [36].

Після порівняння віддано перевагу Visual Studio.

2.4 Висновок за другим розділом

У розділі 2 було проведено аналіз можливостей ігрових рушіїв. В результаті їх порівняння віддано перевагу Unity, бо він простий та надає великий асортимент безкоштовних ассетів в магазині. Інструмент Unity має безліч переваг, які перетворюють його на чудовий засіб розробки ігор. Також було розглянуто та обґрунтовано вибір середовища розробки Visual Studio та мови програмування C#, для реалізації 3D гри для Android. Visual Studio обрано заради зручності, стилізації коду та кількості плагінів. А C# – заради простоти та гарної читабельності.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ ГРИ

3.1 Проектування функціоналу гри

На рисунках 3.1-3.2 наведено діаграми прецедентів, що визначають майбутній функціонал мобільної гри «Cross Way Driving».

На рисунку 3.1 наведено діаграму прецедентів, що демонструє основні дії в сцені магазину.

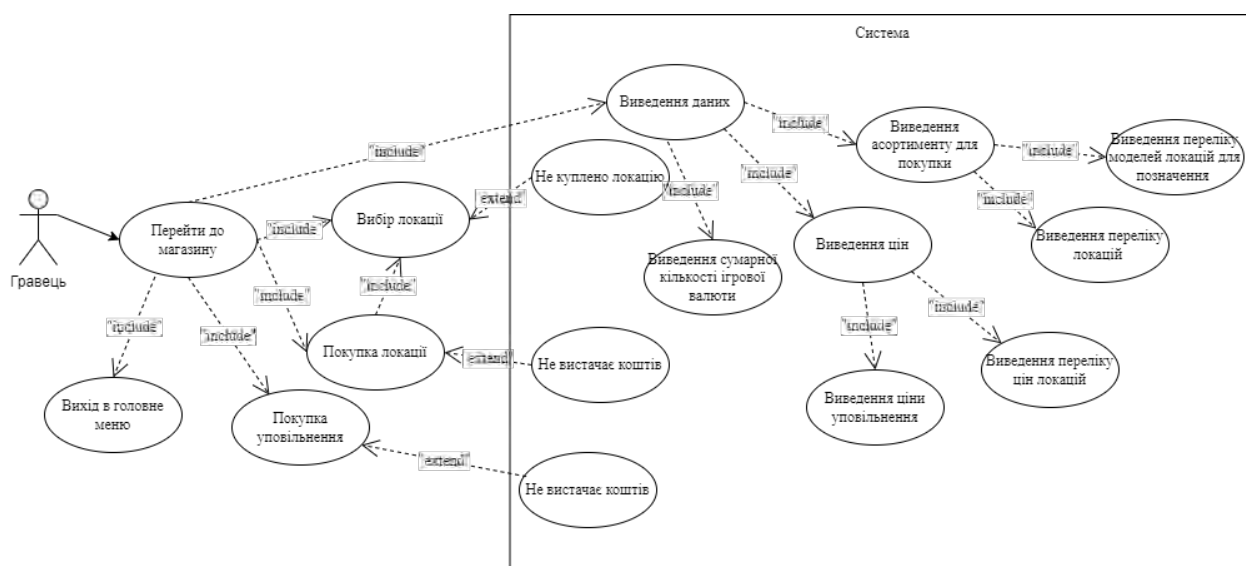


Рисунок 3.1 – Use Case діаграма дій гравця в сцені магазину

Нижче у таблиці 3.1 наведено основні етапи основного, альтернативного та помилкових потоків для гравця.

Таблиця 3.1 – Етапи основного потоку, альтернативних та помилкових потоків для гравця

Основний потік	Альтернативні потоки	Потоки помилок
Гравець бажає виконати дії перейшовши до магазину: вибір, покупка локації, покупка уповільнення та вихід в головне меню.	A1. Вибір локації не відбувається. A2. Покупка локації не здійснюється. A3. Покупка уповільнення не здійснюється.	E1. Не придбано локацію. E2. Недостатньо коштів для покупки локації. E3. Недостатньо коштів для покупки уповільнення.

Далі наведено детальний опис етапів.

Основний виконавець (актор) – гравець.

Потоки подій.

Етапи основного потоку подій:

а) гравець бажає виконати дії перейшовши до магазину:

1) гравець вибирає локацію. Якщо локацію не придбано, то виконується альтернативний потік A1 та виконується потік помилок E1;

2) гравець купує локацію. Після чого може вибрати її для подальшої гри. Якщо недостатньо коштів для покупки, виконується альтернативний потік A2 та потік помилок E2;

3) гравець купує уповільнення. Якщо недостатньо коштів для покупки, виконується альтернативний потік A3 та потік помилок E3;

4) гравець виходить з магазину в головне меню;

5) система виводить дані асортименту для покупки, тобто перелік локацій та їх моделей для позначення. Також вона виводить ціни локацій та уповільнення. Крім того, система виводить сумарну кількість ігрової валюти, що заробив гравець.

Етапи альтернативних потоків:

A1. Вибір локації не відбувається. Перейти до 2 кроку.

A2. Покупка локації не здійснюється. Повернення до початку основного потоку.

A3. Покупка уповільнення не здійснюється. Повернення до початку основного потоку.

Етапи потоків помилок:

E1. Не придбано локацію. Немає можливості вибрати її без придбання.

E2. Недостатньо коштів для покупки локації. Відтворюється звук помилки покупки.

E3. Недостатньо коштів для покупки уповільнення. Відтворюється звук помилки покупки.

На рисунку 3.2 наведено діаграму прецедентів, що демонструє основні дії в головній сцені.

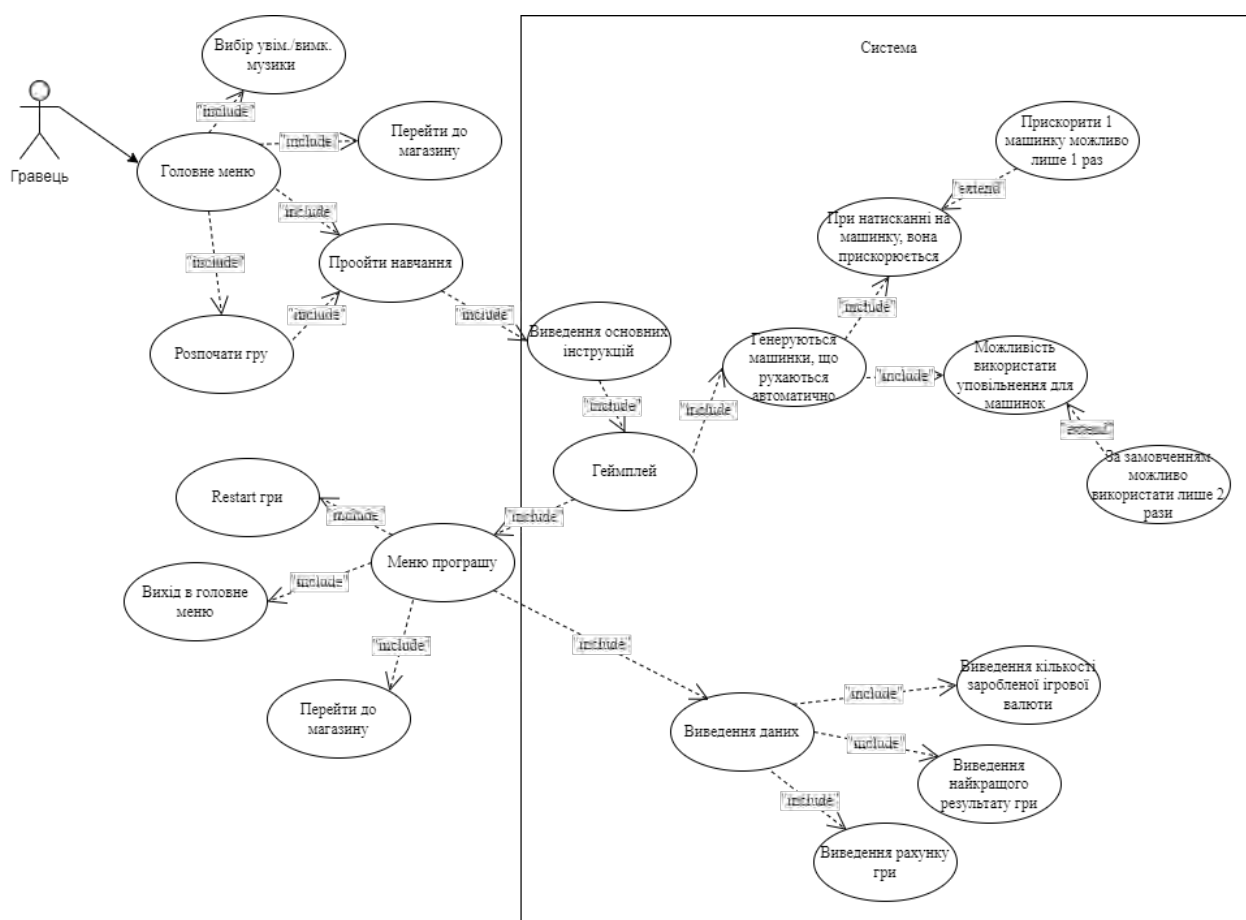


Рисунок 3.2 – Use Case діаграма дій гравця в головній сцені

Нижче у таблиці 3.2 наведено основні етапи основного, альтернативного та помилкових потоків для гравця.

Таблиця 3.2 – Етапи основного потоку, альтернативних та помилкових потоків для гравця

Основний потік	Альтернативні потоки	Потоки помилок
Гравець бажає виконати дії в головному меню: увім./вимк. музику, перехід до магазину, пройти навчання, розпочати гру.	A1. Прискорення не відбувається. A2. Уповільнення машинки не здійснюється.	E1. Прискорення не спрацює. E2. Уповільнення не спрацює.

Далі наведено детальний опис етапів.

Основний виконавець (актор) – гравець.

Потоки подій.

Етапи основного потоку подій:

а) гравець бажає виконати дії в головному меню гри:

1) гравець має можливість увімкнути або вимкнути музику та звуки гри;

2) гравець має можливість перейти та виконати дії в магазині;

3) гравець має можливість пройти навчання після чого розпочати гру;

4) гравець має можливість одразу розпочати гру, якщо навчання вже було пройдено;

5) система виводить основні інструкції щодо проходження гри;

6) система генерує машинки, що рухаються з певною швидкістю та заданим напрямом;

7) при натисканні гравцем машинки, відбувається її прискорення. Якщо машинка була вже прискорена, то виконується альтернативний потік А1 та потік помилок Е1. При натисканні на кнопку уповільнення, відбувається уповільнення машинки в 2 рази після деякого часу швидкість повертається. Якщо використано 2 уповільнення, виконується альтернативний потік А2 та потік помилок Е2;

8) при зіткненні машинок виводиться меню програшу, де гравець має можливість розпочати гру знову, перейти до головного меню, або перейти до магазину;

9) система виводить дані меню програшу: кількість заробленої ігрової валюти, найкращий результат гри та поточний рахунок гри.

Етапи альтернативних потоків:

А1. Прискорення не відбувається. Повернення до кроку 6 основного потоку.

А2. Уповільнення машинки не здійснюється. Повернення до кроку 6 основного потоку.

Етапи потоків помилок:

Е1. Прискорення не спрацьовує. Немає можливості прискорити одну машинку декілька разів.

Е2. Уповільнення не спрацьовує. Блокується кнопка та змінюється зображення на ній на неактивну.

3.2 Проектування локацій гри

В Asset Store було завантажено та імпортовано в гру три ассети «Low poly styled trees», «Simple City Lite» та «SimpleTownLite». З «Low poly styled trees» було використано дерева для локацій. А з двох інших ассетів – будинки, дорога, машинки та інші речі. Спроектвані локації зображено на рисунках 3.3-3.5.



Рисунок 3.3 – Локація Suburb



Рисунок 3.4 – Локація City



Рисунок 3.5 – Локація Downtown

3.3 Опис модулів мобільної гри

В ході роботи над програмою дипломного проєкту було розроблено класи: ButtonsPressing, FadingScreen, MoveCar, WestTurnLeft, WestTurnRight, EastTurnLeft, EastTurnRight, NorthTurnLeft, NorthTurnRight, SouthTurnLeft, SouthTurnRight, CreatingCars, DeleteCar, Accelaration, CollisionCar, CarLights, ScoreShow, ShowCoins, ShowLoseGameMenu, SlowTime, FirstStep, SecondStepStudy, BuySlowTime, BuyNewLocation, SelectLocation, ShowLocation, ExhaustEffectSound, ExplodeEffectSound, CreatingAudio, AudioControl (таблиця 3.3).

Таблиця 3.3 – Опис класів реалізованої гри з визначенням основних функцій

Клас	Опис
ButtonsPressing	Скріпт, що містить у собі функціонал кнопок
FadingScreen	Скріпт ефекту згасання сцени
MoveCar	Скріпт, що контролює рух машинок
WestTurnLeft	Скріпт західного повороту машини ліворуч
WestTurnRight	Скріпт західного повороту машини праворуч
EastTurnLeft	Скріпт східного повороту машини ліворуч
EastTurnRight	Скріпт східного повороту машини праворуч
NorthTurnLeft	Скріпт північного повороту машини ліворуч
NorthTurnRight	Скріпт північного повороту машини праворуч
SouthTurnLeft	Скріпт південного повороту машини ліворуч
SouthTurnRight	Скріпт південного повороту машини праворуч
CreatingCars	Скріпт створення машинок
DeleteCar	Скріпт видалення машинок
Accelaration	Скріпт прискорення машинок
CollisionCar	Скріпт, який відстежує, якщо машинки зіткнулися, то гра завершується
CarLights	Скріпт відповідає за включення та вимикання будь-якого поворотника машинок
ScoreShow	Скріпт, що показує рахунок гравця
ShowCoins	Скріпт, показує монети гравця
ShowLoseGameMenu	Скріпт виведення меню програшу
SlowTime	Скріпт уповільнення машини
FirstStep	Скріпт першого кроку навчання у грі
SecondStepStudy	Скріпт другого кроку навчання у грі
BuySlowTime	Скріпт придбання уповільнення в магазині
BuyNewLocation	Скріпт придбання нової локації у магазині
SelectLocation	Скріпт для вибору локації гри
ShowLocation	Скріпт для відображення вибраної локації гри
ExhaustEffectSound	Скріпт звуку ефекту прискорення машинки під час натискання неї
ExplodeEffectSound	Скріпт звуку ефекту зіткнення машинок, тобто вибуху
CreatingAudio	Скріпт, який відповідає за незникнення звуку під час переходу між сценами
AudioControl	Скріпт контролю аудіо (звуки самої гри та гудків машин)

Нижче будуть наведені методи розроблених класів, що реалізують програмне забезпечення та їх опис (табл. 3.4-3.33). Повний код розроблених класів наведено у Додатку А.

Таблиця 3.4 – Дані та методи класу ButtonsPressing

Реалізація класу	Опис
1	2
public Sprite btn;	При натисканні кнопка втискатиметься (за замовчуванням – одна картинка)
public Sprite btnPressed;	Коли натиснули на кнопку картинка змінилася
public bool music;	Для вибору увім./вимк. музики
private Image image;	Для зміни картинки на кнопках
private float yPosition;	Початкова позиція кнопки
private Transform child;	Додаткове зображення для переміщення
void Start ();	Функція старт, що використовується для ініціалізації змінних
void OnMouseDown ();	Викликається при натисканні на об'єкт
void OnMouseUp ();	Викликається під час відпускання кнопки миші на об'єкті
void OnMouseUpAsButton ();	Спрацьовує, коли натиснули на кнопку і відпустили в районі одного колайдери
IEnumerator loadScene (string scene);	Створення корутини завантаження сцени

Таблиця 3.5 – Дані та методи класу FadingScreen

Реалізація класу	Опис
1	2
public Texture2D fadingTexture;	Зображення fading
private float fadeSpeed = 0.8f;	Швидкість згасання
private int opacityDepth = -1000;	Наскільки чорним стане екран (непрозорість)
private float alpha = 1.0f;	Момент становлення повністю непрозорим екрану (цілком чорний)
private int fadeDirection = -1;	Перехід від чорного
void OnGUI ();	Рисовка затемнення за допомогою текстури
public float BeginFade (int direction);	Повертає швидкість загасання екрану
void OnEnable ();	Перехід від чорного до прозорого екрану

Таблиця 3.6 – Дані та методи класу MoveCar

Реалізація класу	Опис
1	2
public GameObject turnSignal;	Сигнал повороту
private Rigidbody Righb;	Завдяки компоненту все рухатимемо
public float speedCar = 9f;	Швидкість машинки
void Start ();	Функція старт, що використовується для ініціалізації змінних
void FixedUpdate ();	Оновлення прорисовки, що спричиняє рух позиції об'єкта

Таблиця 3.7 – Дані та методи класу WestTurnLeft

Реалізація класу	Опис
1	2
private Rigidbody Righb;	Компонент руху
private float rotationVelocity;	Швидкість обертання машинки
void Start ();	Функція старт, що використовується для ініціалізації змінних
void FixedUpdate ();	Оновлення прорисовки, викликає функцію leftTurn();
void leftTurn ();	Поворот машинки ліворуч

Таблиця 3.8 – Дані та методи класу WestTurnRight

Реалізація класу	Опис
1	2
private Rigidbody Righb;	Компонент руху
private float rotationVelocity;	Швидкість обертання машинки
void Start ();	Функція старт, що використовується для ініціалізації змінних
void FixedUpdate ();	Оновлення прорисовки, викликає функцію rightTurn ();
void rightTurn ();	Поворот машинки праворуч

Таблиця 3.9 – Дані та методи класу EastTurnLeft

Реалізація класу	Опис
1	2
private Rigidbody Righb;	Компонент руху
private float rotationVelocity;	Швидкість обертання машинки
void Start ();	Функція старт, що використовується для ініціалізації змінних
void FixedUpdate ();	Оновлення прорисовки, викликає функцію leftTurn ();
void leftTurn ();	Поворот машинки ліворуч

Таблиця 3.10 – Дані та методи класу EastTurnRight

Реалізація класу	Опис
1	2
private Rigidbody Righb;	Компонент руху
private float rotationVelocity;	Швидкість обертання машинки
void Start ();	Функція старт, що використовується для ініціалізації змінних
void FixedUpdate ();	Оновлення прорисовки, викликає функцію rightTurn ();
void rightTurn ();	Поворот машинки праворуч

Таблиця 3.11 – Дані та методи класу NorthTurnLeft

Реалізація класу	Опис
1	2
private Rigidbody Righb;	Компонент руху
private float rotationVelocity;	Швидкість обертання машинки
void Start ();	Функція старт, що використовується для ініціалізації змінних
void FixedUpdate ();	Оновлення прорисовки, викликає функцію leftTurn ();
void leftTurn ();	Поворот машинки ліворуч

Таблиця 3.12 – Дані та методи класу NorthTurnRight

Реалізація класу	Опис
1	2
private Rigidbody Righb;	Компонент руху
private float rotationVelocity;	Швидкість обертання машинки
void Start ();	Функція старт, що використовується для ініціалізації змінних
void FixedUpdate ();	Оновлення прорисовки, викликає функцію rightTurn ();
void rightTurn ();	Поворот машинки праворуч

Таблиця 3.13 – Дані та методи класу SouthTurnLeft

Реалізація класу	Опис
1	2
private Rigidbody Righb;	Компонент руху
private float rotationVelocity;	Швидкість обертання машинки
void Start ();	Функція старт, що використовується для ініціалізації змінних
void FixedUpdate ();	Оновлення прорисовки, викликає функцію leftTurn ();
void leftTurn ();	Поворот машинки ліворуч

Таблиця 3.14 – Дані та методи класу SouthTurnRight

Реалізація класу	Опис
1	2
private Rigidbody Righb;	Компонент руху
private float rotationVelocity;	Швидкість обертання машинки
void Start ();	Функція старт, що використовується для ініціалізації змінних
void FixedUpdate ();	Оновлення прорисовки, викликає функцію rightTurn ();
void rightTurn ();	Поворот машинки праворуч

Таблиця 3.15 – Дані та методи класу CreatingCars

Реалізація класу	Опис
1	2
private bool mainScene;	Змінна для роботи з головною сценою гри
public GameObject[] cars;	Масив з усіх машинок, які ми можемо створювати та запускати
private float startSpawn;	Час (коли ми завантажили сцену) після якого почне їхати машина з якогось боку
private float waitSpawn;	Час між створенням нової машинки
private int countCars;	Скільки машинок уже проїхало
private bool onceStop;	Змінна для того, щоб ми проробили певну операцію лише 1 раз
void Start ();	Функція старт, що використовується для ініціалізації змінних
void Update ();	Оновлення один раз за кадр із прописаними умовами для роботи гри
IEnumerator westCars ();	Корутина для машинок із західного боку
IEnumerator eastCars ();	Корутина для машинок зі східного боку
IEnumerator northCars ();	Корутина для машинок з північного боку
IEnumerator southCars ();	Корутина для машинок з південного боку

Таблиця 3.16 – Дані та методи класу DeleteCar

Реалізація класу	Опис
1	2
public static int countCars;	Кількість машинок, які успішно проїхали перехрестя
void Start ();	Функція старт, що використовується для ініціалізації змінних (під час запуску завжди 0)
void OnTriggerEnter (Collider other);	Відстежуватиме коли наш другий об'єкт у якого є Rigidbody і BoxCollider, тобто якщо машинка торкнеться блоку межі локації, машинка видаляється

Таблиця 3.17 – Дані та методи класу Acceleration

Реалізація класу	Опис
1	2
public GameObject carExhaust;	Вихлоп, змінна ефекту прискорення
private bool accelerate;	Змінна, яка відображає, якщо ми прискорили хоча б раз машинку, то більше не прискорюємо (прискорити машинку можемо один раз, вдвічі збільшивши швидкість)
void OnMouseDown ();	При натисканні на машинку спрацює ефект прискорення

Таблиця 3.18 – Дані та методи класу CollisionCar

Реалізація класу	Опис
1	2
public GameObject explodeEffect;	Змінна для ефекту вибуху під час зіткнення машинок
public static bool loseGame;	Змінна програшу, до якої можливо достукатися з будь-якого класу (якщо програли, то можливо дізнатися про це з будь-якого класу)
private bool onceStop;	Змінна для того, щоб показати, що ми виконали певну операцію 1 раз
void OnCollisionEnter (Collision other);	При зіткненні машинок зараховується програш та зупиняються машинки

Таблиця 3.19 – Дані та методи класу CarLights

Реалізація класу	Опис
1	2
public int showObject;	Змінна для відображення світла (перший або другий об'єкт, тобто поворотник)
void Start ();	Функція старт, що використовується для ініціалізації змінних
IEnumerator carLights ();	Корутина увімкнення поворотника
IEnumerator sound ();	Корутина звуку поворотника

Таблиця 3.20 – Дані та методи класу ScoreShow

Реалізація класу	Опис
1	2
private Text topResult;	Змінна для рекорду
void OnEnable ();	Виведення на екран рахунку та встановленого рекорду

Таблиця 3.21 – Дані та методи класу ShowCoins

Реалізація класу	Опис
1	2
void OnEnable ();	Виведення на екран усіх монеток

Таблиця 3.22 – Дані та методи класу ShowLoseGameMenu

Реалізація класу	Опис
1	2
private GameObject loseGameMenu;	Змінна меню програшу
private GameObject slowTime;	Змінна уповільнення часу
private bool addOnce;	Змінна, щоб код спрацював 1 раз
void Update ();	Оновлення один раз за кадр

Таблиця 3.23 – Дані та методи класу SlowTime

Реалізація класу	Опис
1	2
private Text countSlow;	Змінна кількості уповільнень
public Sprite active;	Змінна для зміни картинки за наявності та відсутності уповільнення (зміна картини на активну)
public Sprite inactive;	Змінна для зміни картинки за наявності та відсутності уповільнення (зміна картини на неактивну)
void Start ();	Функція старт, що використовується для ініціалізації змінних з перевіркою умов для ефекту уповільнення
void OnMouseDown ();	При натисканні на кнопку сповільнюємо гру
IEnumerator slowCar ();	Корутина для уповільнення часу
void OnDisable ();	Коли переходимо на іншу сцену або оновлюємо сцену, повертаємо звичайний час

Таблиця 3.24 – Дані та методи класу FistStepStudy

Реалізація класу	Опис
1	2
public GameObject secondCar;	Змінна, що відкриває другу машинку
public GameObject carExhaust;	Змінна для прискорення машинки
public Text study;	Змінна для виведення тексту першого кроку
private bool firstStepStudy;	Змінна потрібна, щоб показати, що перший етап пройдено
void Start ();	Функція старт, що використовується для ініціалізації змінних (виведення першої підказки)
void Update ();	Оновлення один раз за кадр із прописаними умовами для руху машинки
void MoveCar ();	Встановлення швидкості машинки
void OnMouseDown ();	Прискорення при натисканні на машинку
void OnDisable ();	При знищенні першої машинки активується друга

Таблиця 3.25 – Дані та методи класу SecondStepStudy

Реалізація класу	Опис
1	2
public Text study;	Змінна для виведення тексту другого кроку
void Start ();	Функція старт, що використовується для ініціалізації змінних (виведення другої підказки)

Продовження таблиці 3.25

1	2
void OnDisable ();	Коли друга машинка знищена, навчання пройдено та завантажується сцена гри

Таблиця 3.26 – Дані та методи класу BuySlowTime

Реалізація класу	Опис
1	2
public AudioClip success;	Змінна для звуку успішної покупки
public AudioClip fail;	Змінна для звуку не успішної покупки
public Text count;	Змінна виведення кількості уповільнень
public Text coins;	Змінна для виведення кількості монеток
void Start ();	Функція старт, що використовується для ініціалізації змінних
void OnMouseUpAsButton ();	Придбання ефекту уповільнення

Таблиця 3.27 – Дані та методи класу BuyNewLocation

Реалізація класу	Опис
1	2
public AudioSource soundObj;	Змінна звуку натискання кнопки покупки
public AudioClip success;	Змінна для звуку успішної покупки
public AudioClip fail;	Змінна для звуку не успішної покупки
public GameObject checkedBtn;	Змінна вибору кнопки
public Sprite uncheckedBtn;	Змінна спрайта необраної локації
public Sprite checkedBtnSprite;	Змінна спрайта обраної локації
public GameObject[] notSelectLocationBtns;	Масив кнопок зі зміною картинок (кнопки вибору)
public GameObject[] btnsToBuy;	Масив кнопок для покупок
public Text coins;	Змінна виведення кількості монеток
private int locationPrice;	Змінна ціни за яку ми купуємо локацію
private string locationName;	Змінна вибору локації при покупці
void Awake ();	Якщо локація відкрита, то ми відображаємо кнопки вибору (викликається до методу старту)
void OnMouseUpAsButton ();	Придбання локації

Таблиця 3.28 – Дані та методи класу SelectLocation

Реалізація класу	Опис
1	2
public GameObject[] notCurrentLocationBtns;	Масив кнопок зі зміною картинок (приховування кнопок вибору)
public Sprite thisOne;	Змінна картинки кнопки вибору локації(галочка)
public Sprite notThisOne;	Змінна картинки кнопки вибору локації (хрестик)
private string nameOfLocation;	Змінна вибору локації при покупці

Продовження таблиці 3.28

1	2
void Start ();	Функція старт, що використовується для ініціалізації змінних за умовою заданих значень за замовчуванням
void OnMouseUpAsButton ();	Перехід між локаціями

Таблиця 3.29 – Дані та методи класу ShowLocation

Реалізація класу	Опис
1	2
public GameObject[] locations;	Масив із трьох локацій
void Start ();	Функція старт, що використовується для ініціалізації змінних з перевіркою вибору локації та умовою заданих значень за замовчуванням

Таблиця 3.30 – Дані та методи класу ExhaustEffectSound

Реалізація класу	Опис
1	2
public AudioClip[] musicClips;	Масив різних звуків прискорення машинки
void Start ();	Функція старт, що використовується для ініціалізації змінних з перевіркою вибору випадкового звуку ефекту прискорення машинок

Таблиця 3.31 – Дані та методи класу ExplodeEffectSound

Реалізація класу	Опис
1	2
public AudioClip[] musicClips;	Масив різних звуків прискорення машинки
void Start ();	Функція старт, що використовується для ініціалізації змінних (програємо звук при зіткненні)

Таблиця 3.32 – Дані та методи класу CreatingAudio

Реалізація класу	Опис
1	2
public GameObject music;	Змінна музика самої гри
void Start ();	Функція старт, що використовується для ініціалізації змінних (звук не (зникає) знищується при переході на інші сцени)

Таблиця 3.33 – Дані та методи класу AudioControl

	Опис
1	2
private bool play;	Змінна, яка показує, чи грає музика
void Start ();	Функція старт, що використовується для ініціалізації змінних
void Update ();	Оновлення один раз за кадр з прописаними умовами для увімкнення або вимкнення музики гри в залежності від того, чи натиснули кнопку
Enumerator carHorn ();	Корутина сигналу машинки

3.4 Алгоритми розробленої гри

Під час розробки гри було реалізовано наступні алгоритми:

- основний алгоритм функціонування розробленої гри;
- алгоритм завантаження фону головної сцени;
- алгоритм зміни стану кнопки;
- алгоритм зміни стану кнопки музики;
- алгоритм створення машинок;
- алгоритм видалення машинок;
- алгоритм сцени study;
- алгоритм сцени game;
- алгоритм увімкнення меню програшу;
- алгоритм сцени shop.

Наведені алгоритми на рисунках 3.6-3.15 було відображено згідно стандарту ISO 5807-85 [38].

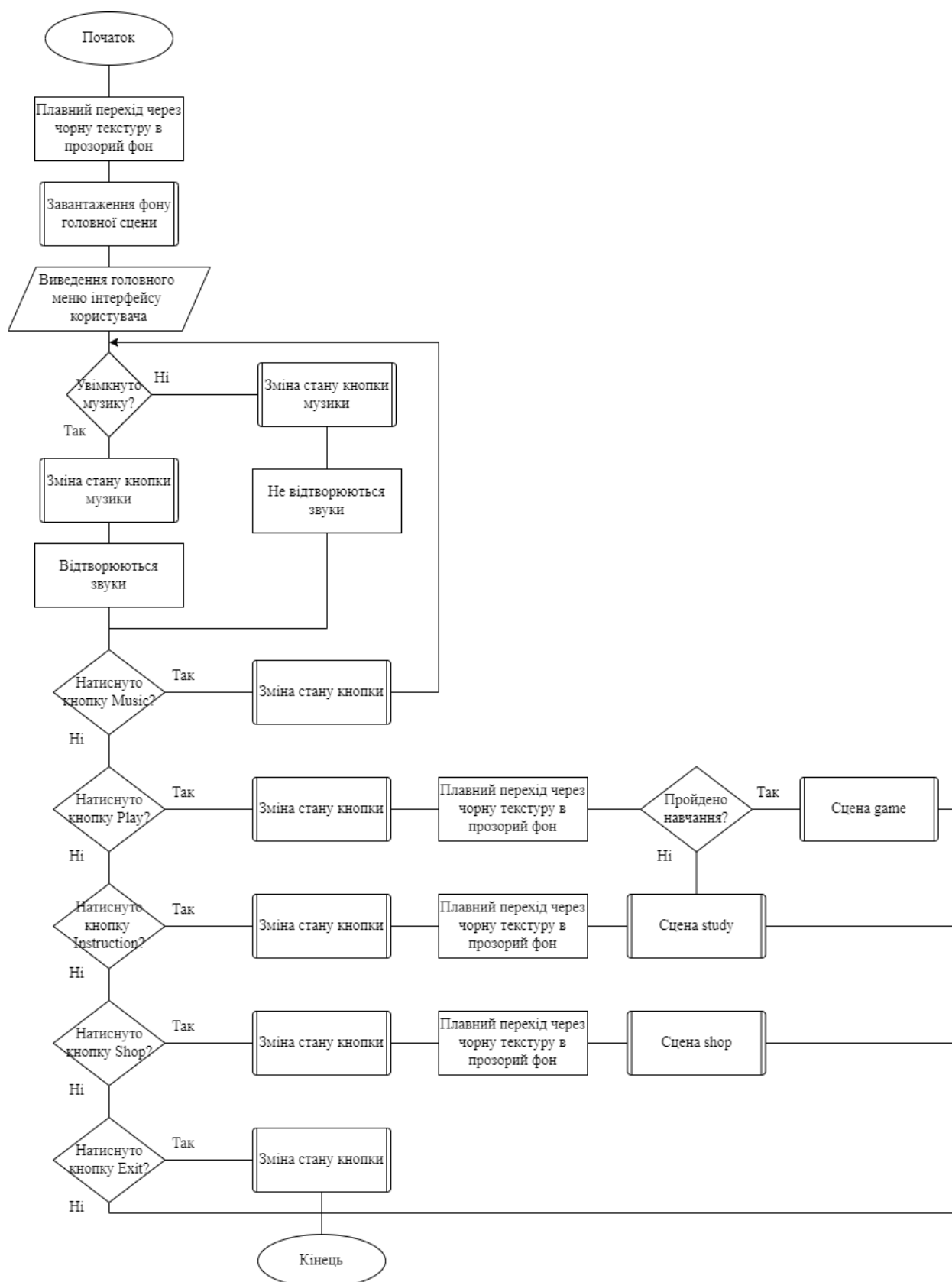


Рисунок 3.6 – Основний алгоритм функціонування гри

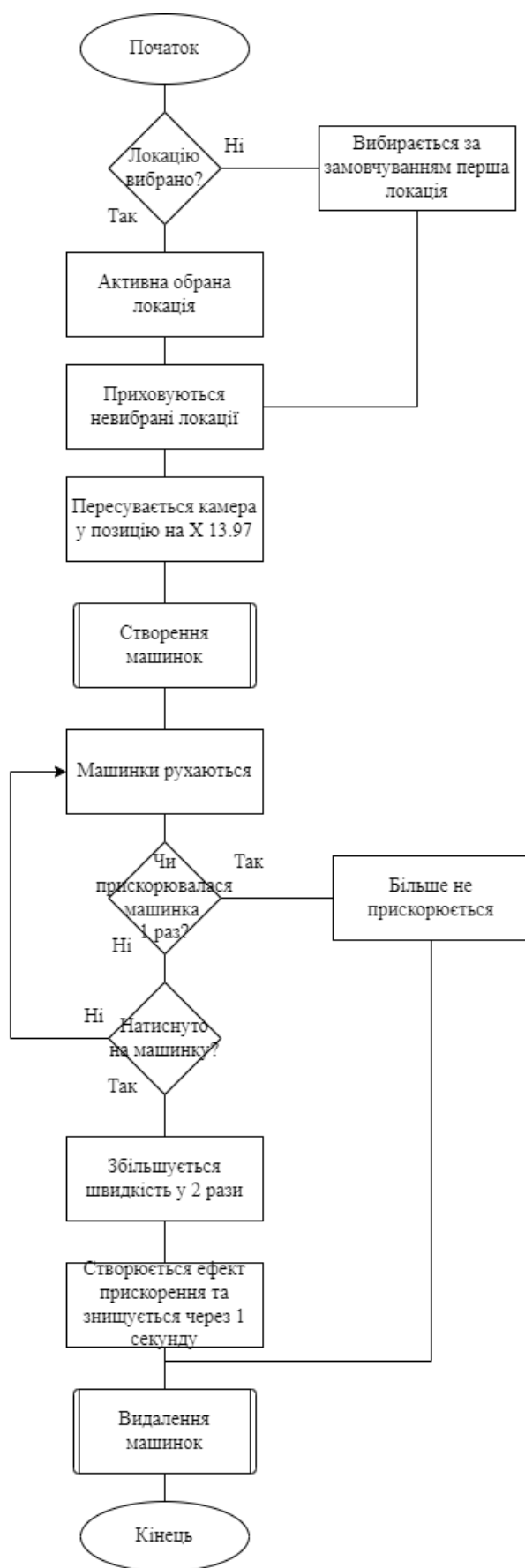


Рисунок 3.7 – Алгоритм завантаження фону головної сцени

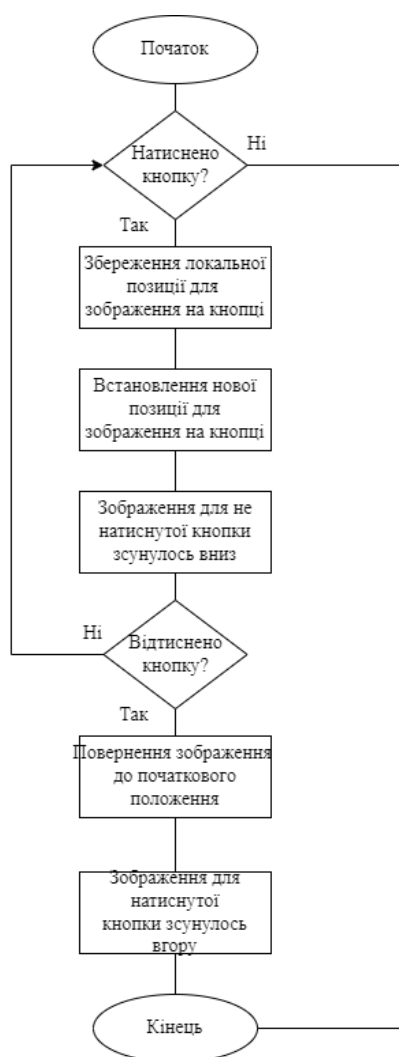


Рисунок 3.8 – Алгоритм зміни стану кнопки



Рисунок 3.9 – Алгоритм зміни стану кнопки музики

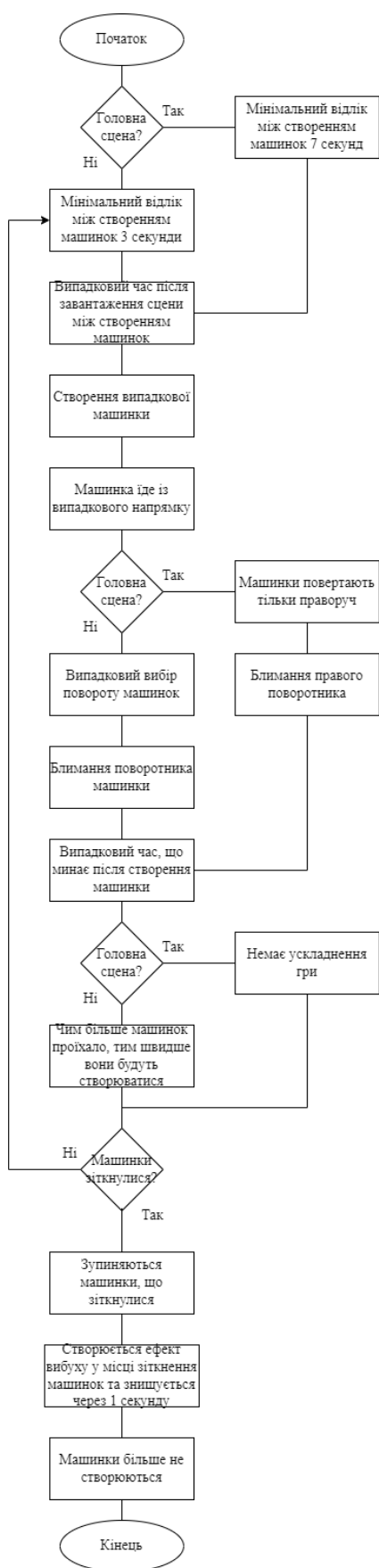


Рисунок 3.10 – Алгоритм створення машинок

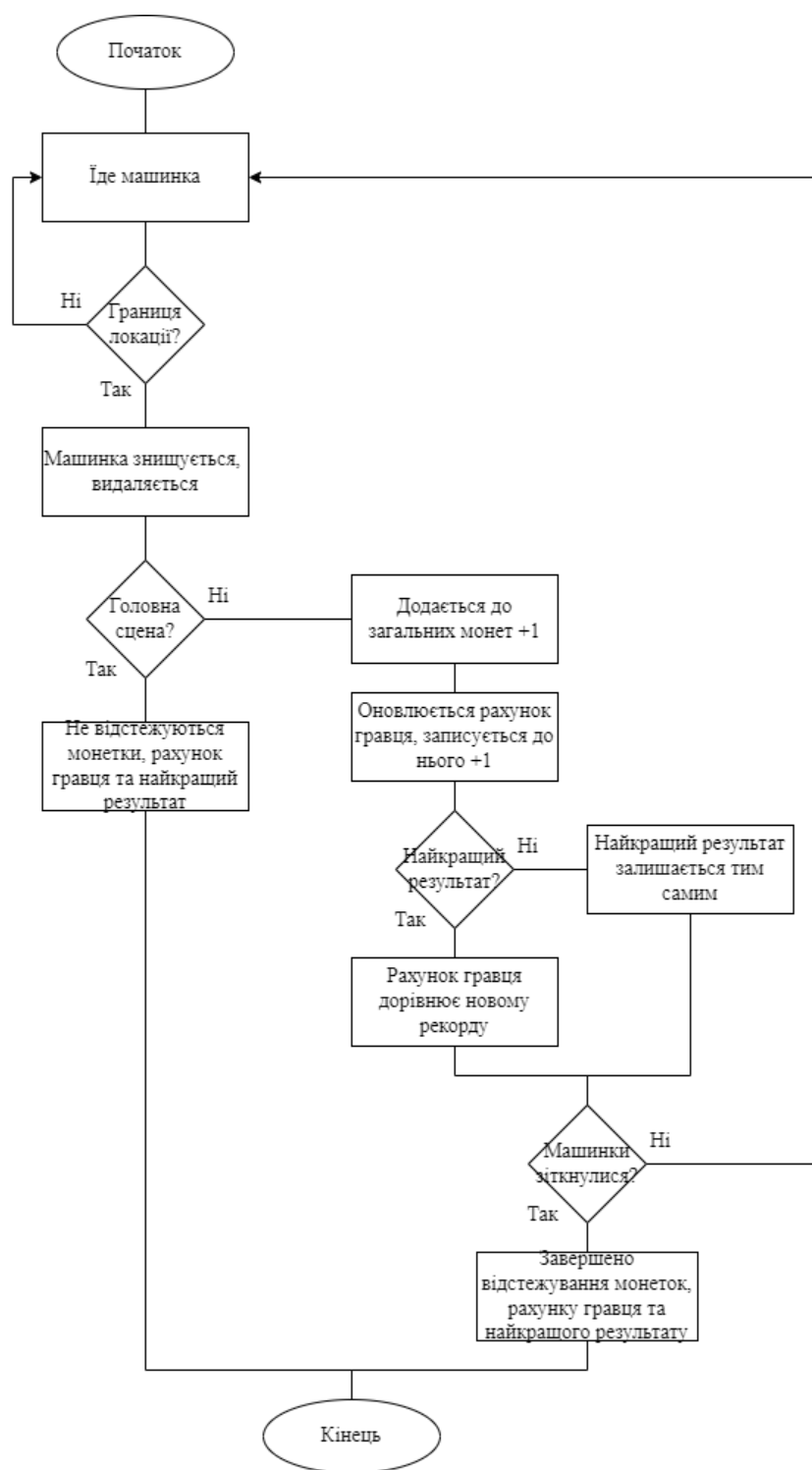


Рисунок 3.11 – Алгоритм видалення машинок



Рисунок 3.12 – Алгоритм сцени study

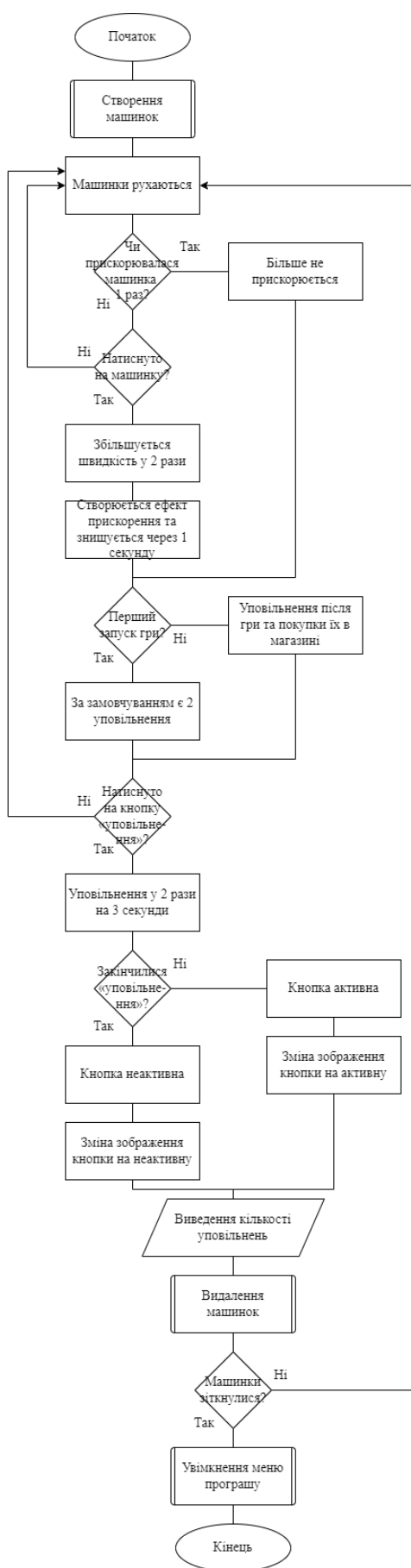


Рисунок 3.13 – Алгоритм сцени game

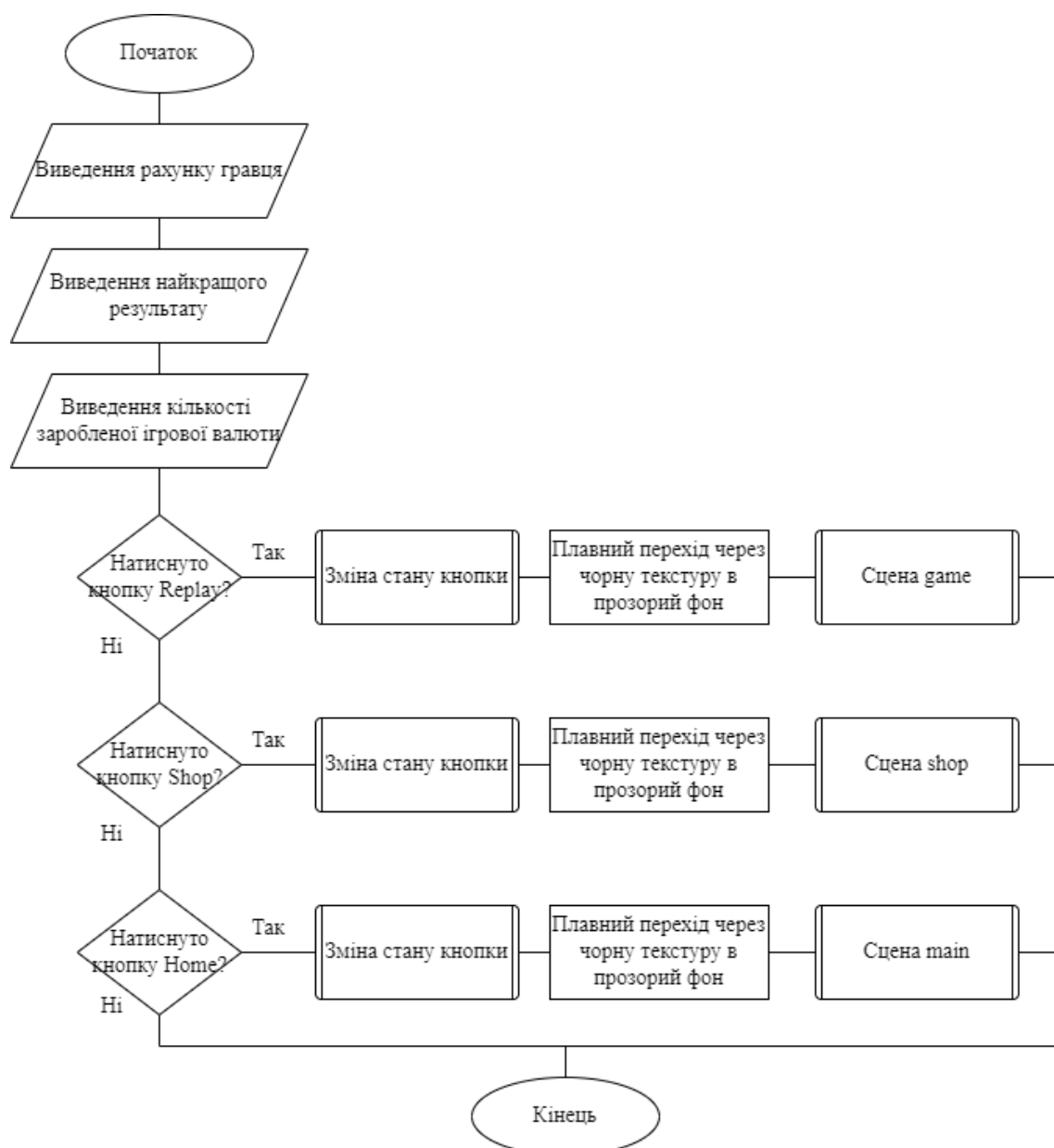


Рисунок 3.14 – Алгоритм увімкнення меню програшу

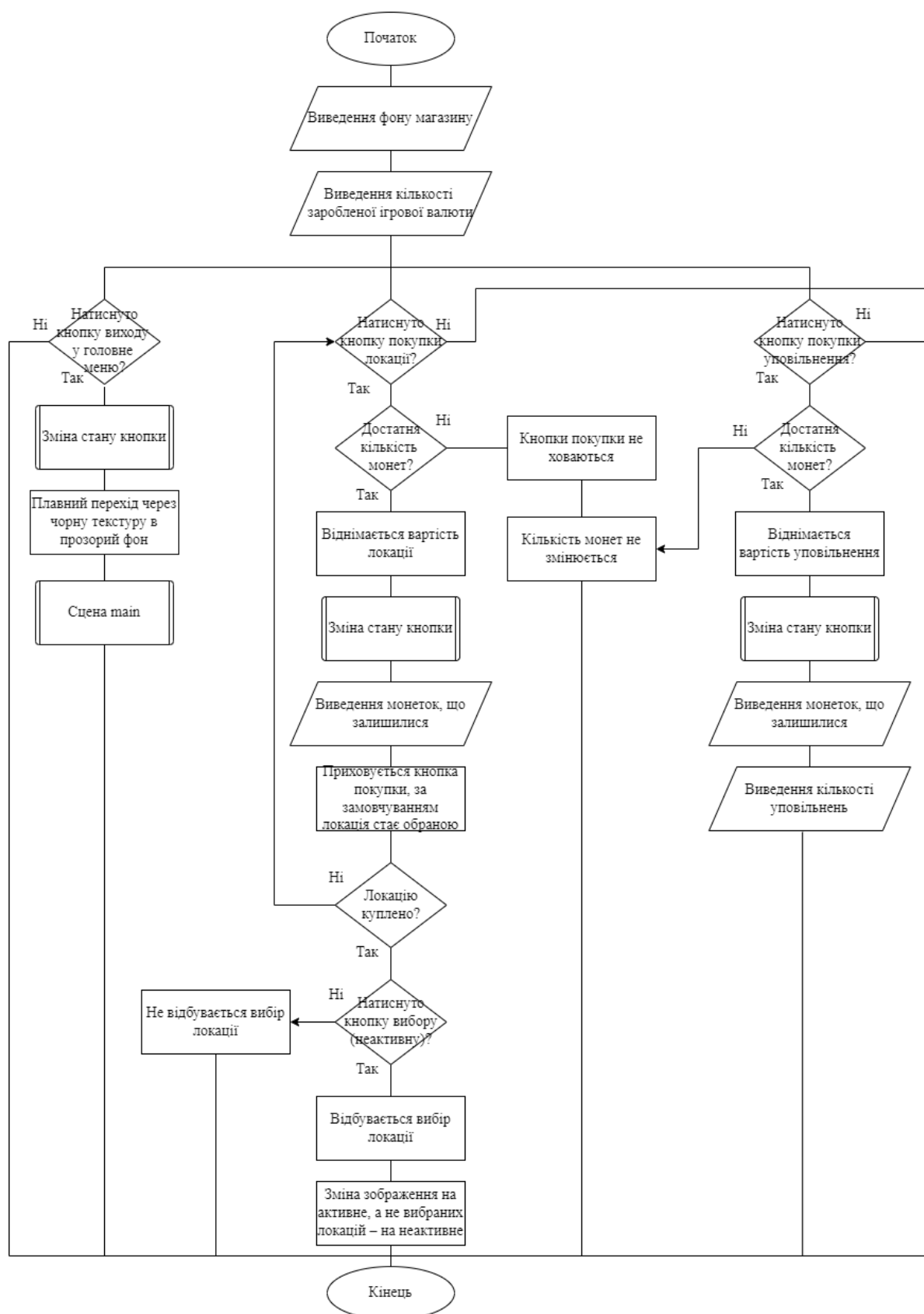


Рисунок 3.15 – Алгоритм сцени shop

На рисунку 3.16 зображено діаграму переходів між сценами мобільної гри. На ній можна побачити сцени та можливі переходи між ними.

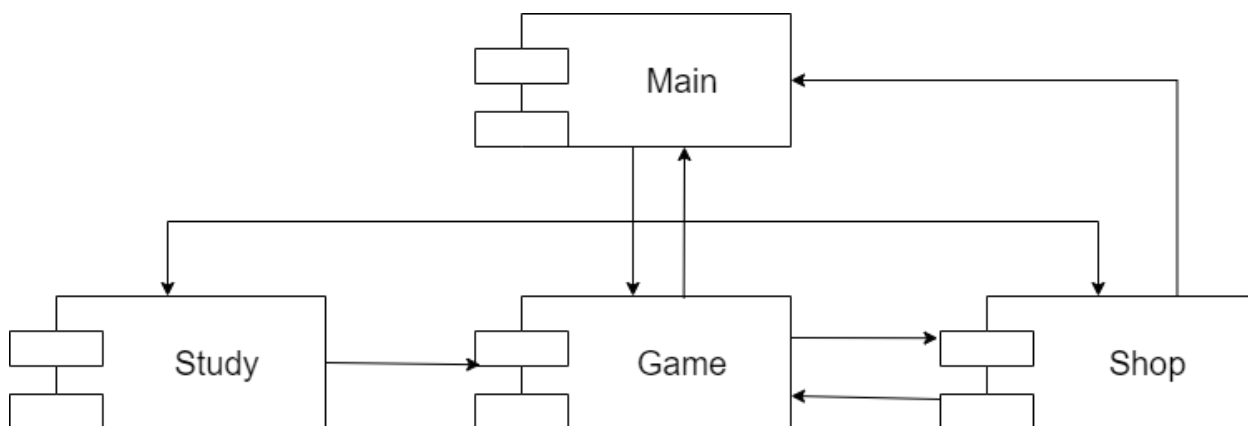


Рисунок 3.16 – Діаграма переходів між сценами гри «Cross Way Driving»

На діаграмі переходів між сценами наведено можливі переходи:

- з головної сцени можливо потрапити на сцени навчання, гри та магазину;
- на сцену навчання можливо потрапити лише з головної сцени;
- на сцену гри можливо потрапити одразу з головної сцени, якщо навчання вже було пройдено, зі сцени навчання та зі сцени магазину при програві;
- на сцену магазину можливо потрапити з головного меню та зі сцени гри при програві;
- на головну сцену можливо потрапити зі сцени гри через меню програву та з магазину.

Такий зв'язок переходів між сценами формує поетапність необхідну для реалізації вимог гравця.

3.5 Висновок за третім розділом

Було спроектовано майбутній функціонал програми, наведено діаграми прецедентів, що демонструють основні дії в головній сцені та сцені магазину. Наведено детальний опис класів гри, її полів та методів. Також спроектовано локації гри. Було наведено основні алгоритми гри та описано переходи між сценами.

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Призначення й умови застосування гри

Робоче ім'я мобільної гри – «Cross Way Driving». Гра призначена для розвитку когнітивних навичок: швидкості реакції, уваги та логічного мислення.

Область застосування – розвиток когнітивних функцій за допомогою відеоігор.

Мобільна гра дає можливість користувачу розвивати швидкість реакції, увагу та логічне мислення в ігровій формі.

Для функціонування гри необхідно виконати умови щодо використання смартфона з наступними апаратними характеристиками:

- операційна система – Android 6.0 чи вище;
- вільний простір пам'яті не менше 100 Мб;
- оснащена мінімум 2 Гб ОЗУ;
- чотириядерний процесор 1.5 ГГц або краще.

Умови застосування для коректної роботи гри:

- використання останніх версій гри;
- без підключення до мережі Інтернет;
- права на читання та запис файлів у директорії, в якій знаходиться програма.

Для отримання повного функціоналу цілком достатньо встановити та запустити гру.

4.2 Основні вимоги до гри

Вимоги до функціональних характеристик мобільної гри «Cross Way Driving»:

- гра має реагувати на дотик до екрану з затримкою не більше 0.15с;

- гра повинна надавати можливість обирати що користувач хоче зробити: перейти до магазину, купити або вибрати локацію, купити уповільнення, пройти навчання, розпочати гру, прискорити або уповільнити машинку, перейти до головного меню та розпочати гру знову;
- програма повинна надавати можливість користувачу після програшу відстежити кількість заробленої ігрової валюти, найкращий результат гри та поточний рахунок гри;
- програма повинна надавати можливість користувачу відстежити придбані локації та кількість уповільнень;
- програма повинна надавати інструкції щодо проходження гри.

4.3 Характеристики гри

Обсяг пам'яті, який займає арк файл – 26.6 Мб.

Обсяг пам'яті, який займає програма – 51.39 Мб.

Програма міститься на зовнішньої або внутрішньої пам'яті.

Програма запускається у разі потреби відпочити, відволіктися, а також для розвитку когнітивних функцій.

Гра виконана за допомогою мови програмування C# в середовищі розробки Visual Studio на ігровому рушії Unity. Проєкт (рис.4.1) містить класи, їх реалізацію, файли ресурсів та проєкту, префаби, шрифти, сцени та матеріали.

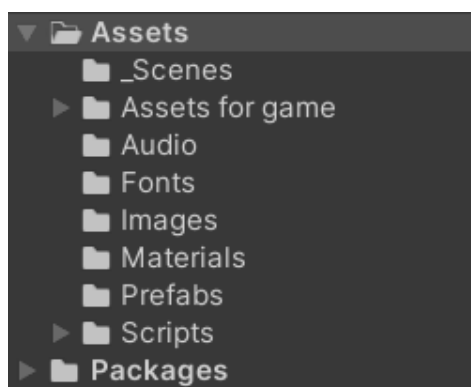


Рисунок 4.1 – Структура проєкту

4.4 Звертання до гри

Для звернення, по-перше, необхідно встановити арк файл.

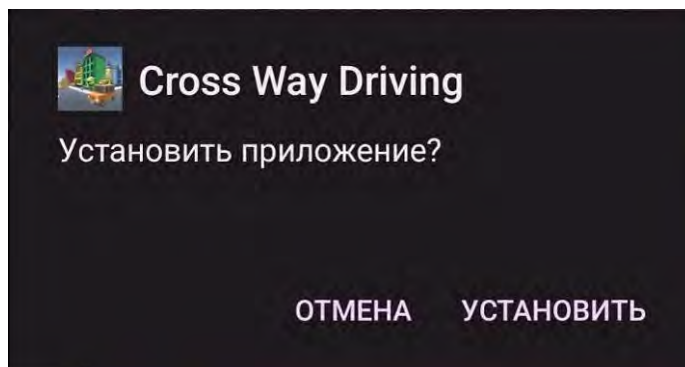


Рисунок 4.2 – Встановлення гри

По-друге, для запуску гри потрібно натиснути на іконку гри (рис.4.2) та дочекатися запуску проєкту. Для повноцінної роботи гри не потрібно наявне підключення до мережі Інтернет.



Рисунок 4.3 – Іконка встановленої гри

4.5 Початкові та вихідні дані

Початкові дані: непройдене навчання, початкова локація, за замовченням в наявності два уповільнення.

Вихідні дані: пройдене навчання, дані рахунку гравця, найкращий результат гри, кількість заробленої ігрової валюти, придбані локації, кількість уповільнень.

Для зберігання даних використовувався вбудований клас `PlayerPrefs`, який працює з такими типами даних: `int`, `float` та `string`.

У Android дані зберігаються на пристрої в `SharedPreferences`. `SharedPreferences` – постійне сховище на платформі Android, яке використовується програмами для зберігання своїх налаштувань та невеликих даних. Це сховище є відносно постійним, користувач може потрапити в налаштування програми та очистити дані, тим самим очистивши всі дані в сховищі [39]. Значення зберігаються як пари: ім'я, значення. Дані `PlayerPrefs` фізично зберігаються в `/data/data/pkg-name/shared_prefs/pkg-name.xml` [40]. Щоб мати можливість бачити `PlayerPrefs`, які Unity зберігає на Android, потрібно мати root-права на пристрої [41]. На рисунку 4.4 прихована папка `shared_prefs`, так як для доступу до даних необхідні Root-права, а на пристрої їх немає.

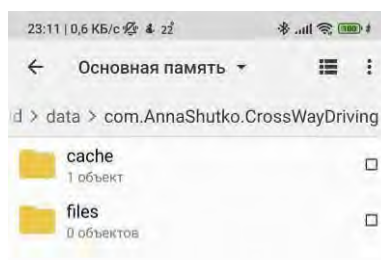


Рисунок 4.4 – Папка, де зберігаються дані гри

Збереження та завантаження даних через `PlayerPrefs` має свої переваги перед іншими видами локального зберігання даних: по-перше простотою роботи, тобто не потрібно працювати з файлами і лізти у файлову систему взагалі, по-друге при роботі з файловою системою, наприклад, на деяких платформах потрібно мати дозвіл на читання та запис даних, для `PlayerPrefs` в цьому немає необхідності він працює на всіх пристроях однаково. Тому `PlayerPrefs` відмінно підходить для зберігання невеликої кількості нескладних даних на пристрої [42].

5 КЕРІВНИЦТВО КОРИСТУВАЧА

5.1 Призначення гри

Робоче ім'я мобільної гри – «Cross Way Driving». Гра призначена для розвитку когнітивних навичок: швидкості реакції, уваги та логічного мислення.

Область застосування – розвиток когнітивних функцій за допомогою відеоігор.

Мобільна гра дає можливість користувачу розвивати швидкість реакції, увагу та логічне мислення в ігровій формі.

5.2 Умови виконання гри

Для функціонування гри необхідно виконати умови щодо використання смартфона з наступними апаратними характеристиками:

- операційна система – Android 6.0 чи вище;
- вільний простір пам'яті не менше 100 Мб;
- оснащена мінімум 2 Гб ОЗУ;
- чотириядерний процесор 1.5 ГГц або краще.

Умови застосування для коректної роботи гри:

- використання останніх версій гри;
- без підключення до мережі Інтернет;
- права на читання та запис файлів у директорії, в якій знаходиться

програма.

Для отримання повного функціоналу цілком достатньо встановити та запустити гру.

5.3 Виконання гри

Щоб розпочати роботу у програмі перш за все потрібно її встановити та запустити натиснувши на іконку гри. Програма готова до роботи.

Гра працює повноцінно без наявності підключення до Інтернету, за ідеальних умов після завантаження з'явиться головне меню гри (рис. 5.1).



Рисунок 5.1 – Головне меню гри

При натисканні на кнопку «проходження навчання» запускається сцена з основними інструкціями щодо проходження гри (5.2).



Рисунок 5.2 – Сцена навчання

В сцені гри генеруються машинки, що рухаються з певною швидкістю та заданим напрямом. Гравець має можливість взаємодіяти з машинками: прискорювати їх, уповільнювати на деякий час (рис. 5.3).



Рисунок 5.3 – Геймплей гри

При зіткненні машинок виводиться меню програшу (рис. 5.4).



Рисунок 5.4 – Меню програшу гри

При натисканні в головному меню, або в меню програшу на кнопку «магазин» гравець потрапляє до сцени магазину, де може придбати локації та

уповільнення (рис. 5.5). При покупці локації з'являється можливість вибору між доступними локаціями.

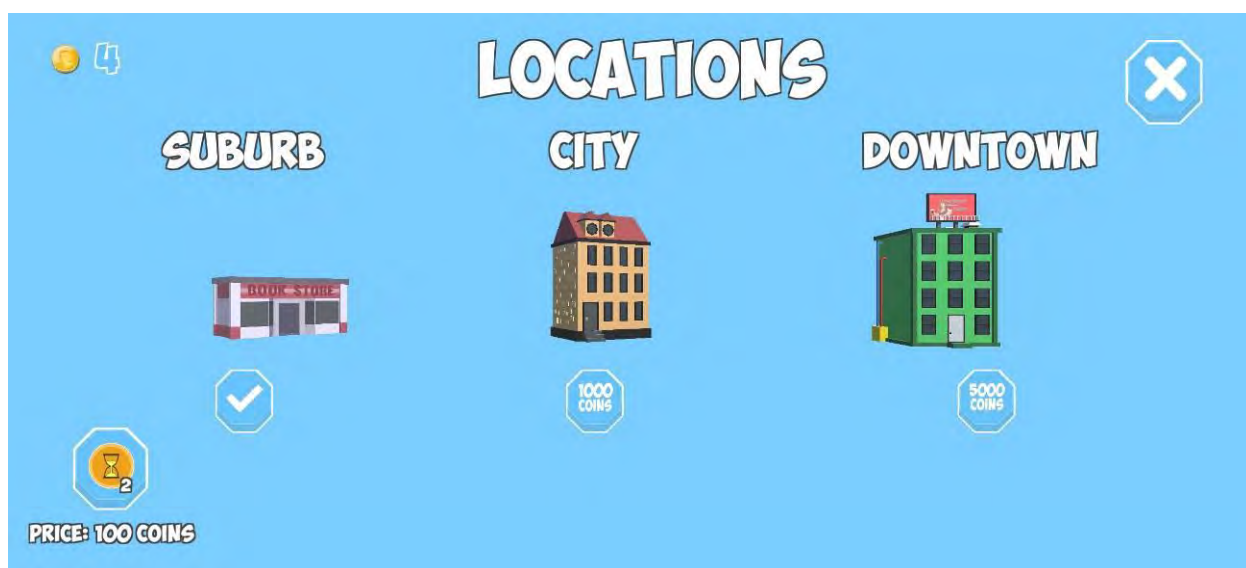


Рисунок 5.5 – Магазин гри

6 ТЕСТУВАННЯ І РЕЗУЛЬТАТИ

6.1 Опис методики проведення тестування

Було проведено ручне тестування функцій гри «Cross Way Driving». Для тестування використано пристрої Mi Pad 4, Redmi 9, POCO F3 та Redmi Note 7 Pro. Перед складанням гри були змінені ціни локацій та уповільнення для показу можливостей гри. Ціна локації «City» змінена на 2 монетки, «Downtown» – 3 монетки, а уповільнення – 1 монетка.

6.2 Результати використання розробленої гри

На головному меню розташовані кнопки керування: «проходження навчання», «запуск гри», «магазин», «керування звуком» та «вихід з гри». На фон додано перехрестя, де рухаються машинки в заданих напрямках. Саме тому машинки спокійно рухаються, не стикаються (рис. 6.1). За замовченням на фоні головного меню відображається локація «Suburb». При натисканні на кнопку керування звуком будуть вмикатися та вимикатися звуки сигналу машин, поворотників, вдалої та невдалої покупки та інші.



Рисунок 6.1 – Головне меню гри за замовченням локація «Suburb»

При натисканні на кнопку «проходження навчання» запускається сцена з основними інструкціями щодо проходження гри. Інструкція складається з двох кроків зображених на рисунках 6.2-6.3.

Крім того, при першому запуску гри при натисканні кнопки «запуск гри» спочатку треба пройти навчання, а далі перейти до гри.



Рисунок 6.2 – Перший крок навчання



Рисунок 6.3 – Другий крок навчання

Генеруються машинки, що рухаються з певною швидкістю та заданим напрямом. Чим більше машинок проїхало, тим швидше вони будуть створюватись, але в головній сцені немає ускладнення гри.

При натисканні гравцем машинки, відбувається її прискорення. Прискорити можливо лише один раз. При натисканні на кнопку «уповільнення», відбувається уповільнення машинки в 2 рази після деякого часу швидкість повертається. При використанні усіх уповільнень кнопка стає неактивною (рис. 6.4-6.5).



Рисунок 6.4 – Геймплей гри з активною кнопкою «уповільнення»



Рисунок 6.5 – Геймплей гри з неактивною кнопкою «уповільнення»

При зіткненні машинок виводиться меню програшу (рис. 6.6), де розташовані кнопки: «розпочати гру знову», «перейти до головного меню», або «перейти до магазину». Також відстежується кількість заробленої ігрової валюти, найкращий результат та поточний рахунок гри.



Рисунок 6.6 – Меню програшу гри

При натисканні в головному меню, або в меню програшу на кнопку «магазин» гравець потрапляє до сцени магазину (рис. 6.7) де реалізовані кнопки: «вибір локації», «покупка локації», «покупка уповільнення» та «вихід в головне меню». Крім того, виводиться кількість заробленої ігрової валюти та назви локацій.

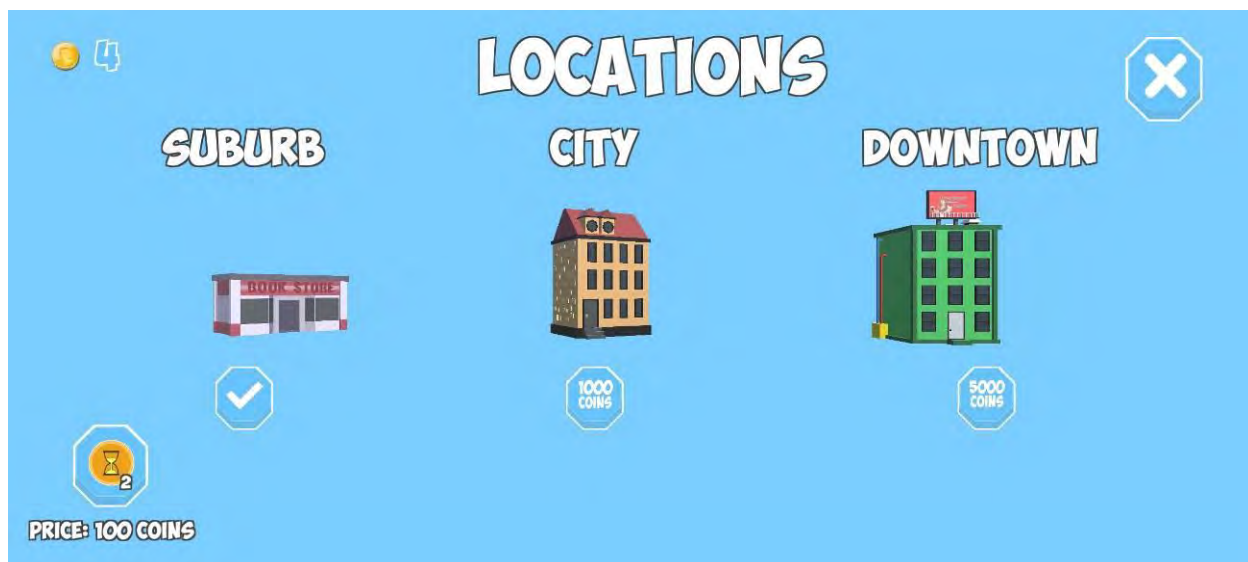


Рисунок 6.7 – Магазин гри

При покупці локації з'являється можливість вибору між доступними локаціями, а при покупці уповільнення змінюється кількість придбаних уповільнень (рис. 6.8). При виборі локації вона відображається на фоні головного меню. На рисунках 6.8-6.9 придбано локацію «City» та відображено обрану локацію на головному меню.

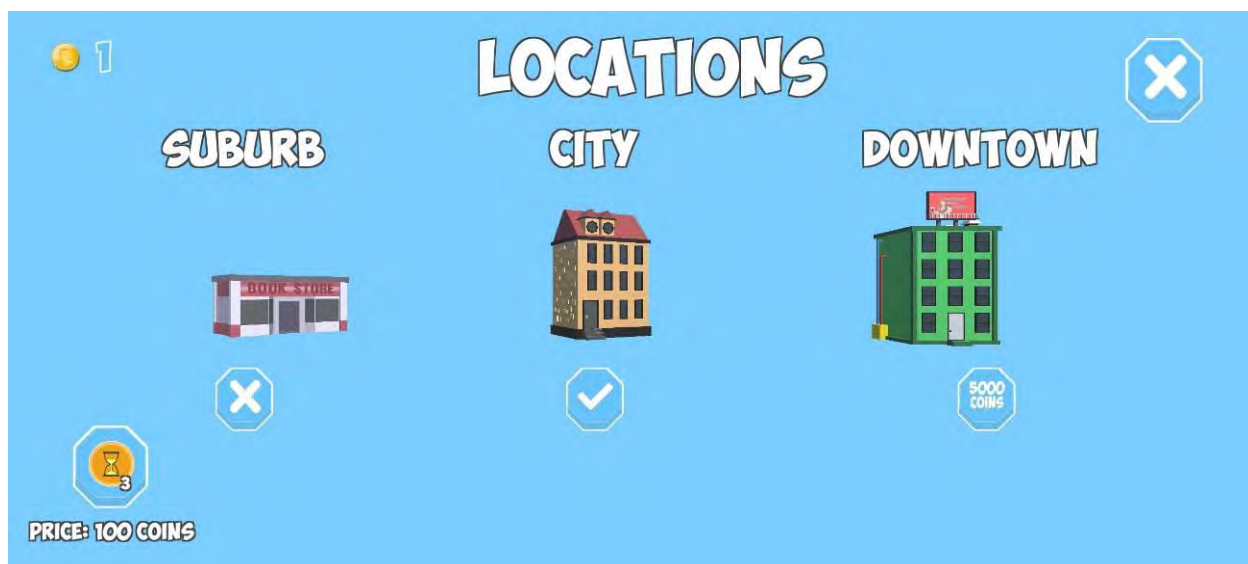


Рисунок 6.8 – Придбано локацію «City» та уповільнення



Рисунок 6.9 – Головне меню з обраною локацією «City»

На рисунках 6.10-6.11 придбано локацію «Downtown» та відображено обрану локацію на головному меню.

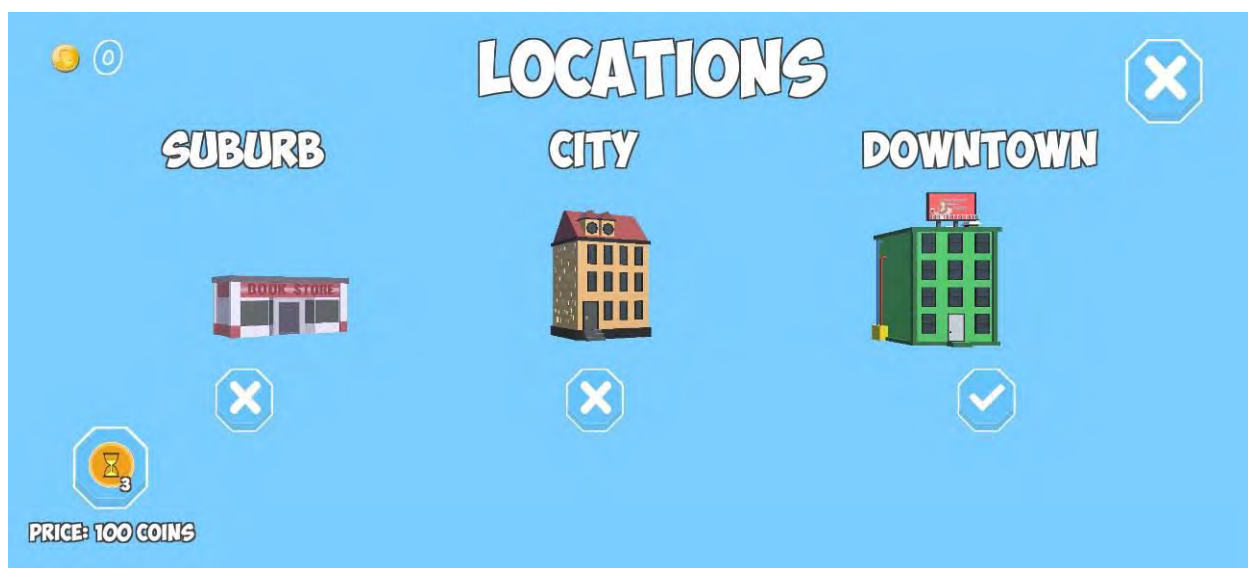


Рисунок 6.10 – Придбано локацію «Downtown»



Рисунок 6.11 – Головне меню з обраною локацією «Downtown»

Результати перевірки функції можна спостерігати на таблиці 6.1. Знайдені дефекти виведені у вигляді таблиці 6.2.

Таблиця 6.1 – Тестування функціональності

Опис функції	Знайдені дефекти на пристроях:			
	Росо F3	Redmi Note 7 Pro	Mi Pad 4	Redmi 9
Купівля елементів	Не знайдено	Не знайдено	Не знайдено	Не знайдено
Відображення локації	Знайдено помилку: номером 1	Не знайдено	Не знайдено	Не знайдено
Прискорення машинки	Не знайдено	Не знайдено	Не знайдено	Не знайдено
Уповільнення машинки	Не знайдено	Не знайдено	Не знайдено	Не знайдено
Навчання	Не знайдено	Не знайдено	Не знайдено	Не знайдено
Коректне відображення поворотників	Не знайдено	Не знайдено	Не знайдено	Не знайдено
Увім./вимк. музики	Не знайдено	Не знайдено	Не знайдено	Не знайдено
Зміна локації	Не знайдено	Не знайдено	Не знайдено	Не знайдено
Виходи машинок за текстуру	Знайдено помилку: номером 2	Знайдено помилку: номером 2	Знайдено помилку: номером 2	Знайдено помилку: номером 2
Клікабельність кнопок	Не знайдено	Не знайдено	Не знайдено	Не знайдено
Коректна зміна активних та неактивних кнопок	Не знайдено	Не знайдено	Не знайдено	Не знайдено
Коректне відображення рахунку/результату/коінів	Не знайдено	Не знайдено	Не знайдено	Не знайдено
Хибні спрацьовування аварій	Не знайдено	Не знайдено	Не знайдено	Не знайдено
Коректне відображення ефектів прискорення та зіткнення машинок	Не знайдено	Не знайдено	Не знайдено	Не знайдено

Таблиця 6.2 – Знайдені дефекти

Номер	Опис	Скріншот
1	При більшій роздільній здатності екрана, камера має більший кут огляду на локацію, через що вона відображає непередбачену текстуру.	Рисунок 6.12
2	При зіткненні машинок, машинка може зачіпити ліхтарний стовп, який не є непрохідною текстурою, тому через нього можуть проїжджати машинки.	Немає можливості зафіксувати у зв'язку з його рідкістю



Рисунок 6.12 – Дефект №1

6.3 Оптимізація гри

Кнопки були зібрані з картинок, а не використовувалися елементи із зображеннями на них, щоб зменшити розмір гри. Крім того, якщо використовувати повне зображення для кожного стану кнопки (рис. 6.13-6.14), то це займає в два рази більше пам'яті, тому для зменшення обсягу потрібної пам'яті було виконана декомпозиція і окремо зберігається фон та картинка, яка накладається зверху (рис. 6.15-6.16).



Рисунок 6.13 – Не натиснута кнопка з зображенням



Рисунок 6.14 – Натиснута кнопка з зображенням



Рисунок 6.15 – Не натиснута кнопка без зображення



Рисунок 6.16 – Натиснута кнопка без зображення

Гра оптимізована під різні екрани за допомогою додавання компонента Aspect Ratio Filter, де Aspect Mode вибрано Height Controls Width. Цей компонент додано до кнопок у головному меню, меню програшу та магазині. Щоб кнопки не розтягувалися залежно від розміру екрана, висота контролюватиме ширину, тобто коли змінюється роздільна здатність екрана, то ширина підлаштовується під висоту кнопок. Кнопки залишаються того ж розміру, але чим більший екран, тим більша відстань між кнопками (рис. 6.17-6.18). Також було розширено локацію для екранів з великою роздільною здатністю.



Рисунок 6.17 – Екран роздільною здатністю 800x480



Рисунок 6.18 – Екран роздільною здатністю 2960x1440

Було оптимізовано скрипти. По можливості було прибрано виклики `GetComponent` або `GameObject.Find` у функціях `void Update ()`; та `void FixedUpdate ()`; – функціях, які викликаються кожен кадр, тому що для того, щоб виконати дані операції потребується трохи пам'яті та продуктивності.

Усі об'єкти, які не рухаються, тобто ті, з якими гравець не взаємодіє, повинні бути позначені статичними. До таких об'єктів відноситься все з чого складаються локації: дороги, дерева, будинки, машини, вуличні ліхтарі та інші речі (рис. 6.19-6.21). Також до них відносяться прозорі куби при зіткненні з якими машинки віддаляються, колайдери дороги, які потрібні, щоб машинки не провалювалися в текстури та колайдери стін, щоб при зіткненні машинок вони не заїхали в текстури будинків. Якщо не робити всі ці об'єкти статичними, можна помітити, що продуктивність гри і її плавність набагато погіршиться.

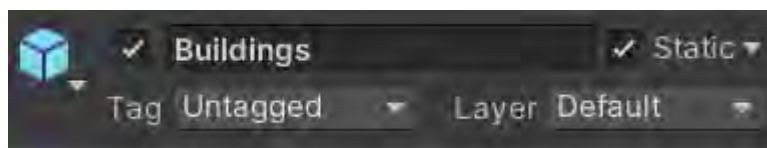


Рисунок 6.19 – Встановлення статичних будинків

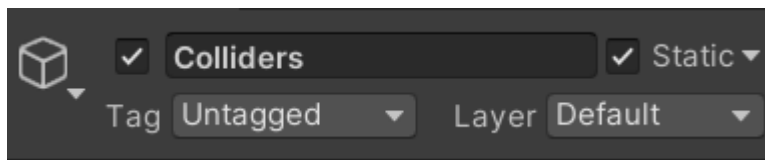


Рисунок 6.20 – Встановлення статичних колайдерів

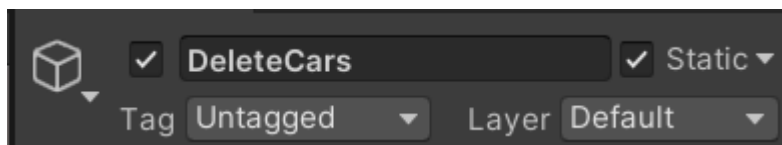


Рисунок 6.21 – Встановлення статичних кубів для видалення машинок

Щоб гра працювала ще плавніше і стабільніше, повороти у вигляді світла машинки було замінено об'єктами (жовті овальні об'єкти). Повороти у вигляді світла більше потребують пам'яті та продуктивності. Було створено матеріал, який має напівпрозорі властивості та картинку світла з переходом від жовтого до прозорого (рис. 6.22-6.23). Далі потрібно було видалити два компоненти світла і замість них створити два циліндри, налаштувати їх розміри та додати створений матеріал.



Рисунок 6.22 – Зображення світла для поворотників

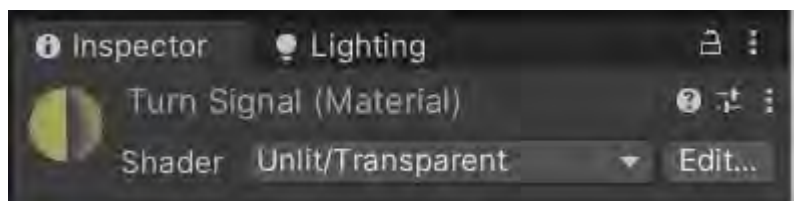


Рисунок 6.23 – Налаштування матеріалу світла для поворотників

І наостанок, є три величезні об'єкти – локації. Вони спочатку закриті і коли будь-яка з них включається з гри, то об'єкт не статичний, а як звичайний і це дуже навантажує гру. Якщо залишити три включені об'єкти, запускати гру і

все вимикати, крім того, який потрібен, то це теж проблематично, тому що перший кадр буде перевантажений і відбудеться виліт з гри. Тому було залишено 3 об'єкти відкритими, але камеру перенесено на позицію X 180 і це дозволяє реалізувати вимкнення всіх локацій, крім однієї на якій гратиме гравець. У скрипті ShowLocation теж зроблено зміни. Якщо локація не дорівнює обраній локації, то приховуємо цю локацію, інакше вибирається за замовчуванням перша локація та ховаються інші. І пересуваємо камеру у потрібну позицію на X 13.97.

Після оптимізації було отримано значення продуктивності гри для різних пристроїв, які наведено в таблиці 6.3.

Таблиця 6.3 – Тестування продуктивності гри

Пристрій	Гра запустилася?	Кількість FPS до оптимізації	Кількість FPS після оптимізації
Росо F3	Так	95	133
Redmi Note 7 Pro	Так	35.3	59.9
Mi Pad 4	Так	35.1	59.8
Redmi 9	Так	36.2	60.2

Нижче на рисунках 6.24-6.27 наведено зображення кількості FPS після оптимізації гри.



Рисунок 6.24 – Кількість FPS у грі на пристрої Росо F3



Рисунок 6.25 – Кількість FPS у грі на пристрої Redmi Note 7 Pro



Рисунок 6.26 – Кількість FPS у грі на пристрої Mi Pad 4



Рисунок 6.27 – Кількість FPS у грі на пристрої Redmi 9

6.4 Висновки за розділом 6

У шостому розділі було обрано перевірки, які гарантуватимуть коректність роботи гри, описано сценарій тестування, наведено діаграму послідовностей.

За результатами тестування на двох різних пристроях Mi Pad 4, Redmi 9, Poco F3 та Redmi Note 7 Pro багів не виявлено, тестування усіх функцій пройшло успішно. Гра стабільна на різних смартфонах Android та версіях Android. Виявлено декілька дефектів, один з них було виправлено.

Завдяки проведеній оптимізації було зменшено розмір гри. Поліпшено її продуктивність, плавність та стабільність. Гра оптимізована під різні екрани.

ВИСНОВКИ

Під час написання пояснювальної записки до дипломної кваліфікаційної роботи бакалавра в першому розділі було проаналізовано дослідження відеоігор з точки зору когнітивного розвитку, вивчення впливу ігор на когнітивний розвиток людини, зв'язок жанрів ігор із різними когнітивними функціями, модулюючими факторами. Було проведено огляд та порівняння аналогів для розвитку когнітивних функцій та було обрано операційну систему Android для якої написана гра, визначено цільову аудиторію. У розділі було розглянуто актуальність проблеми, поняття та основні функції головного мозку. Розглянуто поняття когнітивного розвитку та його областей.

У другому розділі проведено аналіз можливостей ігрових рушіїв, мов програмування та середовищ розробки. В результаті порівнянь віддано перевагу Unity, C# та Visual Studio.

У третьому розділі спроектовано майбутній функціонал програми завдяки наведених діаграм прецедентів, що демонструють основні дії в головній сцені та сцені магазину. Крім того, наведено детальний опис класів гри, її полів та методів. Описано основні алгоритми гри та переходи між сценами.

У четвертому розділі було визначено умови застосування, основні вимоги та характеристики гри. Наведено звертання до гри та визначено вхідні та вихідні дані.

В п'ятому розділі було наведено приклад виконання гри.

В шостому розділі описано методику проведення тестування, наведено результати тестування та оптимізацію гри.

Було розроблено прототип Android гри для розвитку когнітивних функцій: швидкості реакції, уваги та логічного мислення.

Було вирішено усі поставлені задачі. Створена гра відповідає меті розробки, всім вимогам, що були пред'явлені. Передбачено зручний адаптивний інтерфейс користувача з оптимальною панеллю управління.

Практична цінність роботи – можливість розвитку когнітивних функцій.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Piaget Cognitive Development: A Quick Guide: [Electronic resource]. – Access mode: <https://blog.cognifit.com/piaget-cognitive-development/>
2. The Impact of Video Games on Training Surgeons in the 21st Century: [Electronic resource]. – Access mode: <https://jamanetwork.com/journals/jamasurgery/fullarticle/399740>
3. Effects of computer gaming on cognition, brain structure, and function: a critical reflection on existing literature: [Electronic resource]. – Access mode: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6829166/>
4. Games For Seniors – 45 Ways To Help Stimulate The Brain: [Electronic resource]. – Access mode: <https://blog.cognifit.com/games-for-seniors/>
5. Какие бывают все виды видеоигр?: [Электрон. ресурс]. – Режим доступа: <https://www.informatique-mania.com/ru/l'informatique/genres-de-jeux-video/>
6. Esposito, N. A Short and Simple Definition of What a Videogame Is / N. Esposito – Digital Games Research Conference 2005, June 16-20, 2005, Vancouver, British Columbia, Canada, 6 p.
7. Кращі ігрові платформи: [Електрон. ресурс]. – Режим доступу: <https://ukr.kagutech.com/3923350-top-gaming-platforms>
8. 23 Video Game and Online Gaming Statistics, Facts & Trends for 2022: [Electronic resource]. – Access mode: <https://www.cloudwards.net/online-gaming-statistics/>
9. Человеческий мозг: [Электрон. ресурс]. – Режим доступа: <https://www.cognifit.com/ru/brain>
10. Corbetta, M. y Shulman, G. L. (2002). Control of goal-directed and stimulus-driven attention in the brain. *Nat Rev Neurosci*, 3 (3), 201-215 p.
11. Miller, E. K. (2000). The prefrontal cortex and cognitive control. *Nat Rev Neurosci*, 1 (1), 59-65 p.

12. Miller, E. K. y Cohen, J. D. (2001). An integrative theory of prefrontal cortex function. *Annu Rev Neurosci*, 24, 167-202 p.
13. Squire, L.R. (1992) Memory and the hippocampus: a synthesis from findings with rats, monkeys and humans. *Psychol Rev*, 99, pp.195-231 p.
14. Когнитивность и Когнитивистика: [Электрон. ресурс]. – Режим доступа: <https://www.cognifit.com/ru/cognition>
15. Commercial video games and cognitive functions: video game genres and modulating factors of cognitive enhancement: [Electronic resource]. – Access mode: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6996164/>
16. Does Video Gaming Have Impacts on the Brain: Evidence from a Systematic Review: [Electronic resource]. – Access mode: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6826942/>
17. Компьютерные игры и мозг: [Электрон. ресурс]. – Режим доступа: <https://postnauka.ru/video/89851>
18. Pinelle D, Wong N, Stach T. Using genres to customize usability evaluations of video games. In: *Proceedings of the 2008 conference on future play: research, play, share*. New York: ACM; 2008.
19. Apperley TH. Genre and game studies: toward a critical approach to video game genres. *Simul Gaming*. 2006;37(1):6–23. doi: 10.1177/1046878105282278.
20. Greenberg BS, et al. Orientations to video games among gender and age groups. *Simul Gaming*. 2010;41(2):238–259. doi: 10.1177/1046878108319930.
21. Khenissi MA, et al. Relationship between learning styles and genres of games. *Comput Educ*. 2016;101:1–14. doi: 10.1016/j.compedu.2016.05.005.
22. Умные Игры: [Электрон. ресурс]. – Режим доступа: <https://www.cognifit.com/ru/public/games>
23. Игра на благо науки: [Электрон. ресурс]. – Режим доступа: https://pikabu.ru/story/igra_na_bлаго_nauki_4178562
24. Geometry Dash: [Electronic resource]. – Access mode: https://store.steampowered.com/app/322170/Geometry_Dash/?l=ukrainian&curator_c

lanid=27532386

25. NVIDIA, Microsoft, Epic Games, Unity и ведущие разработчики запускают гейминг нового поколения на GDC 2019: [Электрон. ресурс]. – Режим доступа: <https://ru.blogs.nvidia.com/blog/2019/04/03/tracing-support/>

26. Unity3D или Unreal Engine 4: [Электрон. ресурс]. – Режим доступа: <https://stfalcon.com/ru/blog/post/unity3d-vs-unreal-engine-4>

27. Unity vs Unreal Engine for XR Development: Which One Is Better? [2021 Updated]: [Electronic resource]. – Access mode: <https://circuitstream.com/blog/unity-vs-unreal/>

28. Plans and pricing: [Electronic resource]. – Access mode: https://store.unity.com/front-page?check_logged_in=1

29. Unity vs Unreal, What Kind of Game Dev Are You?: [Electronic resource]. – Access mode: <https://www.incredibuild.com/blog/unity-vs-unreal-what-kind-of-game-dev-are-you>

30. Unreal or Unity Engine: Fight for Supremacy: [Electronic resource]. – Access mode: <https://www.gamedesigning.org/engines/unity-vs-unreal/>

31. Дослідження технологій створення освітніх комп'ютерних ігор: [Електрон. ресурс]. – Режим доступу: https://ela.kpi.ua/bitstream/123456789/30992/1/Kostiukevych_magistr.pdf

32. Thorn , A. Mastering Unity Scripting / A. Thorn – Packt Publishing (January 29, 2015), 380 p.

33. 7 языков, на которых можно писать в Unity: [Электрон. ресурс]. – Режим доступа: <https://itproger.com/news/7-yazikov-na-kotorih-mozhno-pisat-v-unity>

34. Bond, J.G. Unity and C#. Gamedev from idea to implementation. 2nd ed / J.G. Bond – Addison-Wesley Professional; 2nd edition (August 20, 2017), 1024 p.

35. What should I prefer C# or Javascript to code in Unity or should I pick some other tool than Unity?: [Electronic resource]. – Access mode: <https://www.quora.com/What-should-I-prefer-C-or-Javascript-to-code-in-Unity-or-should-I-pick-some-other-tool-than-Unity>

36. Difference Between MonoDevelop vs Visual Studio: [Electronic resource]. – Access mode: <https://www.educba.com/monodevelop-vs-visual-studio/>
37. Лучшие IDE для C#-разработчика: [Электрон. ресурс]. – Режим доступа: <https://spb.deveducation.com/blog/luchshie-ide-dlya-c-razrabotchika/#article-block-1>
38. ISO 5807-85: [Electronic resource]. – Access mode: <https://standards.itih.ai/catalog/standards/iso/e1a4e558-03f1-46cd-9ff7-a9abc479294a/iso-5807-1985>
39. SharedPreferences. Сохранение данных в постоянное хранилище Android: [Электрон. ресурс]. – Режим доступа: <https://www.fandroid.info/sharedpreferences-sohranenie-dannyh-v-postoyannoe-hranilishhe-android/>
40. PlayerPrefs: [Electronic resource]. – Access mode: <https://docs.unity3d.com/ru/2019.4/ScriptReference/PlayerPrefs.html>
41. Where is PlayerPrefs stored on Android device?: [Electronic resource]. – Access mode: <https://forum.unity.com/threads/where-is-playerprefs-stored-on-android-device.527849/>
42. Сохранение и загрузка данных в Unity игре: [Электрон. ресурс]. – Режим доступа: <https://unity3dschool.com/sohranenie-i-zagruzka-dannyh-v-unity-igre.html>

ДОДАТОК А
Текст програми

A.1 ButtonsPressing.cs

```

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

public class ButtonsPressing : MonoBehaviour {

    public Sprite btn, btnPressed; // для зміни зображення
    public bool music;

    private Image image;
    private float yPosition;
    private Transform child;

    void Start () {

        image = GetComponent <Image> ();
        child = transform.GetChild (0).transform; // Беру перший дочірній елемент

        if (music) { // працюємо далі, якщо це кнопка music
            if (PlayerPrefs.GetString ("Music") != "no") { // Музика зараз грає, і ми можемо її вимкнути.
                PlayerPrefs-збережені дані під назвою 'Music'!= "no"
                transform.GetChild (0).gameObject.SetActive (true); // відображається зображення за
                замовчуванням
                transform.GetChild (1).gameObject.SetActive (false); // ховається друге
                зображення
            } else {
                transform.GetChild (0).gameObject.SetActive (false);
                transform.GetChild (1).gameObject.SetActive (true);
                child = transform.GetChild (1).transform; // зберігається другий дочірній
                елемент
            }
        }

        void OnMouseDown () { // затиснули кнопку
            image.sprite = btnPressed;
            yPosition = child.localPosition.y; // привласнюю значення кнопки, зберігаю локальну позицію,
            тобто де знаходиться дочірній елемент
            child.localPosition = new Vector3 (child.localPosition.x, 0, child.localPosition.z); //
            встановлюю нову позицію для зображення на кнопці. Ставлю по 0, щоб зображення змістилося вниз
        }

        void OnMouseUp () { // відтиснули кнопку
            image.sprite = btn;
            child.localPosition = new Vector3 (child.localPosition.x, yPosition, child.localPosition.z); //
            повернення у початкове становище
        }

        void OnMouseUpAsButton () {
            if (PlayerPrefs.GetString ("Music") != "no") // якщо музика увімкнена
                GetComponent <AudioSource> ().Play (); // то беремо компонент AudioSource та
                його програємо
            switch (gameObject.name) { // перевірка імені об'єкта, на якому знаходиться скрипт (натискання на
                кнопки)
                case "Play":
                    string scene = PlayerPrefs.HasKey ("Study") ? "game" : "study"; // перехід в іншу
                    сцену ефектом згасання. Роблю все у корутині. Якщо навчання було пройдено, то одразу переходимо в сцену
                    гри, а якщо ні, то проходимо навчання

```

```

        StartCoroutine (loadScene (scene)); // передається назва сцени, яку потрібно
увімкнути
        break;
    case "Replay":
        StartCoroutine (loadScene ("game")); // перезапускаємо сцену
        break;
    case "Home":
        StartCoroutine (loadScene ("main"));
        break;
    case "Instruction":
        StartCoroutine (loadScene ("study"));
        break;
    case "Shop":
        StartCoroutine (loadScene ("shop"));
        break;
    case "CloseGame":
        StartCoroutine (loadScene ("main"));
        break;
    case "Exit":
        Application.Quit();
        break;
    case "Music":
        child.gameObject.SetActive (false); // деактивую об'єкт
        if (PlayerPrefs.GetString ("Music") != "no") { // музика зараз грає і ми можемо її
вимкнути
            PlayerPrefs.SetString ("Music", "no"); // ключу Music встановлюю
значення по
            child = transform.GetChild (1).transform; // встановлюю 2 дочірній
елемент
        } else {
            PlayerPrefs.DeleteKey ("Music"); // вмикаємо музику
            child = transform.GetChild (0).transform; // встановлюю 1 дочірній
елемент
        }
        child.gameObject.SetActive (true); // вмикаємо 2 об'єкт
        break;
    }
}

IEnumerator loadScene (string scene) { // створення корутини завантаження сцени
    float fadeTime = Camera.main.GetComponent <FadingScreen> ().BeginFade (1); // у скіпта
FadingScreen беру функцію BeginFade (1-затемняємо екран) і повертає час переходу
    yield return new WaitForSeconds (fadeTime); // корутина чекає певний час і щось виконує
    SceneManager.LoadScene (scene); // перехід на наступну сцену
}
}

```

A.2 FadingScreen.cs

```

using System.Collections;
using UnityEngine;

public class FadingScreen : MonoBehaviour {

    public Texture2D fadingTexture; // зображення fading

    private float fadeSpeed = 0.8f; // швидкість згасання
    private int opacityDepth = -1000; // наскільки чорним стане екран (непрозорість)
    private float alpha = 1.0f; // момент становлення повністю непрозорого екрана (цілком чорний)
    private int fadeDirection = -1; // перехід від чорного

```

```

void OnGUI () {
    alpha += fadeDirection * fadeSpeed * Time.deltaTime; //Time.deltaTime для плавного
згасання
    alpha = Mathf.Clamp01 (alpha); // значення в межах від 0 до 1 (0-повністю прозорий)

    GUI.color = new Color (GUI.color.r, GUI.color.g, GUI.color.b, alpha); // колір затемнення
    GUI.depth = opacityDepth; // на скільки стане чорним
    GUI.DrawTexture (new Rect (0, 0, Screen.width, Screen.height), fadingTexture); //
текстура, яку будемо малювати. Вказуємо що вона буде повністю на весь екран по x і по y, ширина повністю
весь екран, висота теж весь екран і роблю все за допомогою текстури яку вказали спочатку
}

public float BeginFade (int dir) { // повертає швидкість загасання екрану
    fadeDirection = dir;
    return fadeSpeed;
}

void OnEnable () { // перехід від чорного до прозорого екрану
    BeginFade (-1);
}
}

```

A.3 MoveCar.cs

```

using UnityEngine;

[RequireComponent (typeof (Rigidbody))]// Rigidbody повинен бути на кожній машинці (автоматичне
підключення)
public class MoveCar : MonoBehaviour {

    public GameObject turnSignal;// звук сигналу повороту
    private Rigidbody Righ; // завдяки компоненту будемо все рухати

    public float speedCar = 9f; // швидкість машинки

    void Start () { // привласнюємо значення змінної
        Righ = GetComponent <Rigidbody> ();
    }

    void FixedUpdate () {
        Righ.MovePosition (transform.position - transform.forward * speedCar *
Time.fixedDeltaTime);//рухаю позицію об'єкта (віднімаю рух уперед, щоб рух був прямо)
    }
}

```

A.4 WestTurnLeft.cs

```

using System.Collections;
using UnityEngine;

[RequireComponent (typeof (CarLights))] // підключення скрипта блимання поворотниками машинки
[RequireComponent (typeof (MoveCar))] // підключення скрипта руху машинки
public class WestTurnLeft : MonoBehaviour {

    private Rigidbody Righ; // компонент руху
    private float rotationVelocity; // швидкість обертання машинки

    void Start () {

```

```

Rigb = GetComponent <Rigidbody> ();//привласнюємо значення
      GetComponent <CarLights> ().showObject = 1;
    }

    void FixedUpdate () { // оскільки ми працюємо з фізикою, то краще FixedUpdate, ніж Update
leftTurn (); //виклик функції повороту
    }

    void leftTurn() {
float carRotation = Mathf.Floor (transform.eulerAngles.y); // округляє до меншого числа
      if (transform.localPosition.x > -7f && carRotation != 180f) { // перевірка моменту
повороту і коли потрібно зупиняємо машинки при повороті
          if (carRotation >= 179f && carRotation <= 189f) { // якщо велика швидкість, ми
не встигаємо обробити 180 і машинка повертає сильно
              transform.localRotation = Quaternion.Euler (new Vector3 (0, 180f, 0)); //
зупиняємо машинку на 180
          }
          return;
      }
      rotationVelocity = GetComponent <MoveCar> ().speedCar * -8.57f; // встановлюю швидкість для
повороту машинки (цифра з мінусом, тому що нам потрібен поворот вліво)
      Quaternion deltaRotation = Quaternion.Euler (new Vector3 (0, rotationVelocity, 0) *
Time.fixedDeltaTime); // вектор обертання машинки по у з плавним обертанням
      Rigb.MoveRotation (Rigb.rotation * deltaRotation); // рухаю його обертання
    }
  }
}

```

A.5 WestTurnRight.cs

```

using System.Collections;
using UnityEngine;

[RequireComponent (typeof (CarLights))]
[RequireComponent (typeof (MoveCar))]
public class WestTurnRight : MonoBehaviour {

    private Rigidbody Rigb; // компонент руху
    private float rotationVelocity; // швидкість обертання машинки

    void Start () {
Rigb = GetComponent <Rigidbody> (); //привласнюємо значення
      GetComponent <CarLights> ().showObject = 0;
    }

    void FixedUpdate () {
rightTurn ();
    }

    void rightTurn () {
float carRotation = Mathf.Floor (transform.eulerAngles.y); // округляє до меншого числа
      if (transform.localPosition.x > -15f && carRotation != 0f) { // перевірка моменту повороту і коли
потрібно зупиняємо машинку при повороті
          if (carRotation >= -1f && carRotation <= 9f) { // якщо велика швидкість, ми не встигаємо
обробити 0 і машинка повертає сильно (4) (-1 для додаткового обертання)
              transform.localRotation = Quaternion.Euler (new Vector3 (0, 0f, 0)); // зупиняємо машинку на 0
          }
          return;
      }
      rotationVelocity = GetComponent <MoveCar> ().speedCar * 8.57f; // встановлюю швидкість для
повороту машинки (цифра з плюсом, тому що нам потрібен поворот вправо)
      Quaternion deltaRotation = Quaternion.Euler (new Vector3 (0, rotationVelocity, 0) *
Time.fixedDeltaTime); // вектор обертання машинки по у з плавним обертанням
    }
  }
}

```

```

        Righb.MoveRotation (Righb.rotation * deltaRotation);// рухаю його обертання
    }
}

}

```

A.6 EastTurnLeft.cs

```

using System.Collections;
using UnityEngine;

[RequireComponent (typeof (CarLights))]
[RequireComponent (typeof (MoveCar))]
public class EastTurnLeft : MonoBehaviour {

    private Rigidbody Righb;// компонент руху
    private float rotationVelocity;// швидкість обертання машинки

    void Start () {
        Righb = GetComponent <Rigidbody> ();//привласнюємо значення
        GetComponent <CarLights> ().showObject = 1;// вибір увімкненого поворотника (ліво)
    }

    void FixedUpdate () {
        leftTurn ();
    }

    void leftTurn () {
        float carRotation = Mathf.Floor (transform.eulerAngles.y);// округляє до меншого числа
        if (transform.localPosition.x < -2.5f && carRotation != 0f) { // перевірка моменту повороту і коли
            потрібно зупиняємо машинку при повороті
            if (carRotation >= -1f && carRotation <= 9f) { // якщо велика швидкість, ми не встигаємо
                обробити 0 і машинка сильно повертає
                transform.localRotation = Quaternion.Euler (new Vector3 (0, 0f, 0)); // зупиняємо машинку на 0
                return;
            }
            rotationVelocity = GetComponent <MoveCar> ().speedCar * -8.57f; // встановлюю швидкість для
            повороту машинки (цифра з мінусом, тому що нам потрібен поворот вліво)
            Quaternion deltaRotation = Quaternion.Euler (new Vector3 (0, rotationVelocity, 0) *
            Time.fixedDeltaTime); // вектор обертання машинки по y з плавним обертанням
            Righb.MoveRotation (Righb.rotation * deltaRotation); // рухаю його обертання
        }
    }

}

```

A.7 EastTurnRight.cs

```

using System.Collections;
using UnityEngine;

[RequireComponent (typeof (CarLights))]
[RequireComponent (typeof (MoveCar))]
public class EastTurnRight : MonoBehaviour {

    private Rigidbody Righb; // компонент руху
    private float rotationVelocity; // швидкість обертання машинки

    void Start () {
        Righb = GetComponent <Rigidbody> (); // привласнюємо значення
    }
}

```

```

GetComponent <CarLights> ().showObject = 0;
}

void FixedUpdate () {
rightTurn();
}

void rightTurn () {
float carRotation = Mathf.Floor (transform.eulerAngles.y); // округляє до меншого числа
if (transform.localPosition.x < 6f && carRotation != 180f) { // перевірка моменту повороту і коли
    потрібно зупиняємо машинку при повороті
    if (carRotation >= 180f && carRotation <= 189f) { // якщо велика швидкість, ми не встигаємо
        обробити 180 і машинка сильно повертає
        transform.localRotation = Quaternion.Euler (new Vector3 (0, 180f, 0)); // зупиняємо машинку на
        180
        return;
    }
    rotationVelocity = GetComponent <MoveCar> ().speedCar * 8.57f; // встановлюю швидкість для
    повороту машинки (цифра з плюсом, тому що нам потрібен поворот вправо)
    Quaternion deltaRotation = Quaternion.Euler (new Vector3 (0, rotationVelocity, 0) *
    Time.fixedDeltaTime); // вектор обертання машинки по у з плавним обертанням
    Rigg.MoveRotation (Rigg.rotation * deltaRotation); // рухаю його обертання
}
}
}

```

A.8 NorthTurnLeft.cs

```

using System.Collections;
using UnityEngine;

[RequireComponent (typeof (CarLights))]
[RequireComponent (typeof (MoveCar))]
public class NorthTurnLeft : MonoBehaviour {

    private Rigidbody Rigg; // компонент руху
    private float rotationVelocity; // швидкість обертання машинки

    void Start () {
        Rigg = GetComponent <Rigidbody> (); // привласнюємо значення
        GetComponent <CarLights> ().showObject = 1;
    }

    void FixedUpdate () {
        leftTurn ();
    }

    void leftTurn () {
        float carRotation = Mathf.Floor (transform.eulerAngles.y); // округляє до меншого числа
        if (transform.localPosition.z < 9f && carRotation != 270f) { // перевірка моменту повороту і коли
            потрібно зупиняємо машинку при повороті
            if (carRotation >= 269f && carRotation <= 279f) { // якщо велика швидкість, ми не встигаємо
                обробити 0 і машинка сильно повертає
                transform.localRotation = Quaternion.Euler (new Vector3 (0, 270f, 0)); // зупиняємо машинку на
                270
                return;
            }
            rotationVelocity = GetComponent <MoveCar> ().speedCar * -8.57f; // встановлюю швидкість для
            повороту машинки (цифра з мінусом, тому що нам потрібен поворот вліво)
            Quaternion deltaRotation = Quaternion.Euler (new Vector3 (0, rotationVelocity, 0) *
            Time.fixedDeltaTime); // вектор обертання машинки по у з плавним обертанням

```

```

        Righb.MoveRotation (Righb.rotation * deltaRotation);// рухаю його обертання
    }
}

}

```

A.9 NorthTurnRight.cs

```

using System.Collections;
using UnityEngine;

[RequireComponent (typeof (CarLights))]
[RequireComponent (typeof (MoveCar))]
public class NorthTurnRight : MonoBehaviour {

    private Rigidbody Righb; // компонент руху
    private float rotationVelocity;// швидкість обертання машинки

    void Start () {
        Righb = GetComponent<Rigidbody>();// привласнюємо значення
        GetComponent <CarLights> ().showObject = 0;
    }

    void FixedUpdate () {
        rightTurn ();
    }

    void rightTurn() {
        float carRotation = Mathf.Floor (transform.eulerAngles.y);// округляє до меншого числа
        if (transform.localPosition.z < 17f && carRotation != 90f) { // перевірка моменту повороту і коли
            потрібно зупиняємо машинку при повороті
            if (carRotation >= 89f && carRotation <= 99f) { // якщо велика швидкість, ми не встигаємо
                обробити 90 і машинка сильно повертає
                transform.localRotation = Quaternion.Euler (new Vector3 (0, 90f, 0)); // зупиняємо машинку на
                90
                return;
            }
            rotationVelocity = GetComponent <MoveCar> ().speedCar * 8.57f; // встановлюю швидкість для
            повороту машинки (цифра з плюсом, тому що нам потрібен поворот вправо)
            Quaternion deltaRotation = Quaternion.Euler (new Vector3 (0, rotationVelocity, 0) *
            Time.fixedDeltaTime); // вектор обертання машинки по у з плавним обертанням
            Righb.MoveRotation (Righb.rotation * deltaRotation); // рухаю його обертання
        }
    }

}

```

A.10 SouthTurnLeft.cs

```

using System.Collections;
using UnityEngine;

[RequireComponent (typeof (CarLights))]
[RequireComponent (typeof (MoveCar))]
public class SouthTurnLeft : MonoBehaviour {

    private Rigidbody Righb; // компонент руху
    private float rotationVelocity;// швидкість обертання машинки

    void Start () {

```

```

Rigidbody = GetComponent<Rigidbody>();// привласнюємо значення
GetComponent<CarLights>().showObject = 1;
}

void FixedUpdate () {
    leftTurn ();
}

void leftTurn () {
    float carRotation = Mathf.Floor (transform.eulerAngles.y);// округляє до меншого числа
    if (transform.localPosition.z > 4.8f && carRotation != 90f) {// перевірка моменту повороту і коли
        потрібно зупиняємо машинку при повороті
        if (carRotation >= 89f && carRotation <= 99f) {// якщо велика швидкість, ми не встигаємо
            обробити 90 і машинка сильно повертає
            transform.localRotation = Quaternion.Euler (new Vector3 (0, 90f, 0));
            return;
        }
        rotationVelocity = GetComponent<MoveCar>().speedCar * -8.57f;// встановлюю швидкість для
        повороту машинки (цифра з мінусом, тому що нам потрібен поворот вліво)
        Quaternion deltaRotation = Quaternion.Euler (new Vector3 (0, rotationVelocity, 0) *
        Time.fixedDeltaTime);// вектор обертання машинки по у з плавним обертанням
        Rigidbody.MoveRotation (Rigidbody.rotation * deltaRotation);// рухаю його обертання
    }
}
}

```

A.11 SouthTurnRight.cs

```

using System.Collections;
using UnityEngine;

[RequireComponent (typeof (CarLights))]
[RequireComponent (typeof (MoveCar))]
public class SouthTurnRight : MonoBehaviour {

    private Rigidbody Rigidbody; // компонент руху
    private float rotationVelocity;// швидкість обертання машинки

    void Start () {
        Rigidbody = GetComponent<Rigidbody>();// привласнюємо значення
        GetComponent<CarLights>().showObject = 0;
    }

    void FixedUpdate () {
        rightTurn ();
    }

    void rightTurn () {
        float carRotation = Mathf.Floor(transform.eulerAngles.y);// округляє до меншого числа
        if (transform.localPosition.z > -3.64f && carRotation != 270f) {// перевірка моменту повороту і коли
            потрібно зупиняємо машинку при повороті
            if (carRotation >= 269f && carRotation <= 279f) {// якщо велика швидкість, ми не встигаємо
                обробити 270 і машинка сильно повертає
                transform.localRotation = Quaternion.Euler (new Vector3 (0, 270f, 0));// зупиняємо машинку на
                270
                return;
            }
            rotationVelocity = GetComponent<MoveCar>().speedCar * 8.57f;// встановлюю швидкість для
            повороту машинки (цифра з плюсом, тому що нам потрібен поворот вправо)
            Quaternion deltaRotation = Quaternion.Euler (new Vector3 (0, rotationVelocity, 0) *
            Time.fixedDeltaTime);// вектор обертання машинки по у з плавним обертанням

```

```

        Righb.MoveRotation (Righb.rotation * deltaRotation); // рухаю його обертання
    }
}

}

```

A.12 CreatingCars.cs

```

using System.Collections;
using UnityEngine;

public class CreatingCars : MonoBehaviour {

    [SerializeField] // щоб mainScene була видна в інспекторі, хоч і private
    private bool mainScene; // змінна для роботи з головною сценою гри
    public GameObject[] cars; // масив з усіх машинок, які ми можемо створювати та запускати
    private float startSpawn = 0.5f, waitSpawn; // перша змінна – час (коли ми завантажили сцену)
    після якого почне їхати машина з будь-якої сторони. Для кожного боку різний час. Друга змінна – час між
    створенням нової машинки
    private int countCars = 0; // при створенні машинки змінна збільшується на 1 (скільки машинок
    вже проїхало)

    private bool onceStop; // для того, щоб ми виконали певну операцію лише 1 раз

    void Start () {
        StartCoroutine (westCars ()); // стартую машинки для кожної сторони
        StartCoroutine (eastCars ());
        StartCoroutine (northCars ());
        StartCoroutine (southCars ());

        waitSpawn = mainScene ? 7f : 3f; // мінімальний відлік з 3 секунд (для головної сцени 7
    секунд)

        CollisionCar.loseGame = false; // машинки ще не зіткнулися, для перезапуску гри
    }

    void Update () { // чим більше машинок проїхало, тим швидше вони будуть створюватися
        if (!mainScene) { // у головній сцені не буде ускладнення гри
            if (countCars > 40)
                waitSpawn = 1f;
            else if (countCars > 30)
                waitSpawn = 1.5f;
            else if (countCars > 20)
                waitSpawn = 2f;
        }

        if (CollisionCar.loseGame && !onceStop) { // ми програли і машинки більше не
    створюємо
            StopAllCoroutines ();
            onceStop = true;
        }
    }

    IEnumerator westCars () { // корутина для машинок із західного боку
        yield return new WaitForSeconds (Random.Range (startSpawn, startSpawn + 4.5f)); // час,
        який має пройти між створенням машинок (у машинок з різних боків унікальний час). Попадається випадкове
        число.

        while (true) {
            GameObject carInst = Instantiate (cars[Random.Range (0, cars.Length)], new
            Vector3 (-83.8f, 0, 2.7f), // створюємо машинку та привласнюємо її змінній. Вибираємо випадкову машинку із
            масиву. Вказую позицію машини.

```

```

Quaternion.Euler (new Vector3 (0, -90f, 0))) as GameObject; //обертання
машинки під час створення
        countCars++;
        int rand = mainScene ? 2 : Random.Range (0, 4); // випадковий вибір того, що
машинка повертатиме ліворуч, праворуч або їхатиме прямо (якщо у нас головна сцена, то ми завжди повертаємо
праворуч)
        switch (rand) {
        case 1:
            carInst.AddComponent <WestTurnLeft> (); //додаю новий компонент
повороту вліво
            break;
        case 2:
            carInst.AddComponent <WestTurnRight> (); //додаю новий компонент
повороту вправо
            break;
        }
        yield return new WaitForSeconds (Random.Range (waitSpawn, waitSpawn + 5f)); //
випадковий час, який минає після створення машинки
    }
}

IEnumerator eastCars () { // корутина для машинок зі східного боку
yield return new WaitForSeconds (Random.Range (startSpawn, startSpawn + 4.5f));
while (true) {
    GameObject carInst = Instantiate (cars[Random.Range (0, cars.Length)], new
Vector3 (30f, 0, 11.08f),
        Quaternion.Euler (new Vector3 (0, 90f, 0))) as GameObject;
    countCars++;
    int rand = mainScene ? 2 : Random.Range (0, 4);
    switch (rand) {
    case 1:
        carInst.AddComponent <EastTurnLeft> (); //додаю новий компонент
повороту вліво
        break;
    case 2:
        carInst.AddComponent <EastTurnRight> (); //додаю новий компонент
повороту вправо
        break;
    }
    yield return new WaitForSeconds (Random.Range (waitSpawn, waitSpawn + 5f));
}
}

IEnumerator northCars () { // корутина для машинок із північного боку
yield return new WaitForSeconds (Random.Range (startSpawn, startSpawn + 4.5f));
while (true) {
    GameObject carInst = Instantiate (cars[Random.Range (0, cars.Length)], new
Vector3 (-9.1f, 0, 64.4f),
        Quaternion.Euler (new Vector3 (0, 0f, 0))) as GameObject;
    countCars++;
    int rand = mainScene ? 2 : Random.Range (0, 4);
    switch (rand) {
    case 1:
        carInst.AddComponent <NorthTurnLeft> (); //додаю новий компонент
повороту вліво
        break;
    case 2:
        carInst.AddComponent <NorthTurnRight> (); //додаю новий компонент
повороту вправо
        break;
    }
    yield return new WaitForSeconds (Random.Range (waitSpawn, waitSpawn + 5f));
}
}

```

```

    }

    IEnumerator southCars () { // корутина для машинок із південного боку
    yield return new WaitForSeconds (Random.Range (startSpawn, startSpawn + 4.5f));
        while (true) {
            GameObject carInst = Instantiate (cars[Random.Range (0, cars.Length)], new
Vector3 (-0.32f, 0, -27.22f),
            Quaternion.Euler (new Vector3 (0, 180f, 0))) as GameObject;
            countCars++;
            int rand = mainScene ? 2 : Random.Range (0, 4);
            switch (rand) {
            case 1:
                carInst.AddComponent <SouthTurnLeft> (); //додаю новий компонент
повороту вліво
                break;
            case 2:
                carInst.AddComponent <SouthTurnRight> (); //додаю новий компонент
повороту вправо
                break;
            }
            yield return new WaitForSeconds (Random.Range (waitSpawn, waitSpawn + 5f));
        }
    }
}

```

A.13 DeleteCar.cs

```

using UnityEngine;
using UnityEngine.SceneManagement;

public class DeleteCar : MonoBehaviour {

    public static int countCars; // кількість машинок, які успішно проїхали наше перехрестя

    void Start () {
        countCars = 0; // під час запуску завжди 0
    }

    void OnTriggerEnter (Collider other) { // буде відстежувати коли наш другий об'єкт у якого є
Rigidbody і BoxCollider, тобто якщо машинка торкнеться блоку межі локації, машинка видаляється
        if (other.tag == "Player") { // якщо наш об'єкт машинка
            if (!CollisionCar.loseGame) { // якщо програли, то вже не записуємо +1
                countCars++;
                string scene = SceneManager.GetActiveScene().name;
                if (scene != "main")
                {
                    PlayerPrefs.SetInt("Coins", PlayerPrefs.GetInt("Coins") + 1); //записуємо монетки, щоразу як
видаляється машинка до загальних монет +1
                }
            }
            Destroy (other.gameObject); // то ми її знищуємо
        }
    }
}

```

A.14 AccelerationCar.cs

```

using UnityEngine;

```

```

[RequireComponent (typeof (MoveCar))]
public class AccelerationCar : MonoBehaviour {

    public GameObject carExhaust;// вихлоп, змінна ефекту прискорення
    private bool accelerate = false;// змінна, яка відображає, якщо ми прискорили хоча б раз
    машинку, то більше не прискорюємо (прискорити машинку можемо один раз в 2 рази збільшивши швидкість)

    void OnMouseDown () { // при натисканні на машинку спрацює метод
        if (!accelerate) { // дорівнює false, як при старті машинки
            GetComponent <MoveCar> ().speedCar *= 2f; // то збільшувати швидкість у 2 рази
            accelerate = true; // змінюю стан змінної

            GameObject ex = Instantiate (carExhaust,
                new Vector3 (gameObject.transform.position.x, 0.2f,
gameObject.transform.position.z),
                Quaternion.Euler (new Vector3 (90, 0, 0))) as GameObject; // створюємо
ефект прискорення на позиції нашої машинки, але вище
            Destroy (ex, 1f); // знищуємо об'єкт через 1 секунду
        }
    }
}

```

A.15 CollisionCar.cs

```

using UnityEngine;

[RequireComponent (typeof (MoveCar))]
public class CollisionCar : MonoBehaviour {

    public GameObject explodeEffect; // змінна для ефекту вибуху під час зіткнення машинок
    (префаб вибуху)
    public static bool loseGame = false; // змінна до якої можна достукатися з будь-якого класу (якщо
    програли, то можемо дізнатися про це з будь-якого класу)
    private bool onceStop; // для того, щоб ми виконали певну операцію лише 1 раз

    void OnCollisionEnter (Collision other) { // коли зіткнемося машинками (колайдери без тригера,
    тобто тверді об'єкти)
        if (other.gameObject.tag == "Player" && !onceStop) { // якщо ми зіткнулися з машинкою і
        щоб це спрацювало 1 раз
            loseGame = true; // ми програли
            onceStop = true; // більше не входимо у перевірку
            gameObject.GetComponent <MoveCar> ().speedCar = 0f; // зупиняю машинку
            other.gameObject.GetComponent <MoveCar> ().speedCar = 0f; // теж саме для
другої машинки
            gameObject.GetComponent <Rigidbody> ().AddRelativeForce (Vector3.forward * -
1000); // додали удар

            // Вибух
            if (gameObject.transform.position.x < other.gameObject.transform.position.x) { // не
важливо яка позиція далі, головне, щоб був 1 вибух, а не кілька
                Vector3 pos = Vector3.Lerp (gameObject.transform.position,
other.transform.position, 0.5f); // вибух саме там, де машинки зіткнулися (знаходимо середнє значення між
векторами)
                Instantiate (explodeEffect, new Vector3 (pos.x, 2.7f, pos.z),
Quaternion.identity); // створюємо вибух, все одно в який бік
            }
        }
    }
}

```

A.16 CarLights.cs

```
using System.Collections;
using UnityEngine;

public class CarLights : MonoBehaviour {

    public int showObject;// показуємо світло (перший чи другий об'єкт, тобто поворотник)

    void Start () {
        StartCoroutine (carLights ());//стартую корутину поворотника
        if (PlayerPrefs.GetString ("Music") != "no")// якщо музика увімкнена
            StartCoroutine (sound ());//стартую корутину звуку у машинки, яка повертає
    }

    IEnumerator carLights () {// корутина увімкнення поворотника
        yield return new WaitForSeconds (0.2f);// чекаю пів секунди
        GameObject light = gameObject.transform.GetChild (showObject).gameObject;//записую
значення світла
        while (true) {// блимаємо світлом
            light.SetActive (!light.activeSelf);
            yield return new WaitForSeconds (0.5f);// інтервал блимання
        }
    }

    IEnumerator sound () {
        GameObject s = Instantiate (GetComponent <MoveCar> ().turnSignal, new Vector3 (0,0,0),
Quaternion.identity) as GameObject;// позиція спрацьовування звуку
        yield return new WaitForSeconds (1.3f);// чекаємо
        Destroy (s.gameObject);// знищуємо об'єкт
    }

}
```

A.17 ScoreShow.cs

```
using UnityEngine;
using UnityEngine.UI;

public class ScoreShow : MonoBehaviour {

    [SerializeField]
    private Text topResult;// змінна для рекорду

    void OnEnable () {// коли текст стане видимим, буде відкрит
        GetComponent <Text> ().text = "Score: " + DeleteCar.countCars.ToString ();//коли
машинка видаляється, оновлюється рахунок
        PlayerPrefs.SetInt ("CarsPassed", PlayerPrefs.GetInt ("CarsPassed") + DeleteCar.countCars);

        if (PlayerPrefs.GetInt ("Score") < DeleteCar.countCars) {// якщо у нас зараз топовий результат
            PlayerPrefs.SetInt ("Score", DeleteCar.countCars);// рахунок дорівнює новому
рекорду
            topResult.text = "Top: " + DeleteCar.countCars.ToString ();//то ми записуємо
топовий результат
        }
        else
            topResult.text = "Top: " + PlayerPrefs.GetInt ("Score").ToString ();//якщо немає
рекорду
    }

}
```

A.18 ShowCoins.cs

```
using UnityEngine;
using UnityEngine.UI;

public class ShowCoins : MonoBehaviour {

    void OnEnable () {
        GetComponent <Text>().text = PlayerPrefs.GetInt ("Coins").ToString ();//показує всі
монетки
    }

}
```

A.19 ShowLoseGameMenu.cs

```
using UnityEngine;

public class ShowLoseGameMenu : MonoBehaviour {

    [SerializeField]
    private GameObject loseGameMenu, slowTime;// loseGameMenu – змінна меню програшу.
slowTime – уповільнення часу
    private bool addOnce;// змінна, щоб код було виконано лише 1 раз

    void Update () {
        if (CollisionCar.loseGame && !addOnce) {// якщо ми програли
            addOnce = true;
            slowTime.SetActive (false);// ховаю кнопку уповільнення
            loseGameMenu.SetActive (true);// відкриваємо меню програшу
        }
    }

}
```

A.20 SlowTime.cs

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

public class SlowTime : MonoBehaviour {

    private Text countSlow;// кількість уповільнень
    public Sprite active, inactive;// змінна для зміни зображення за наявності та відсутності
уповільнення

    void Start () {
        countSlow = gameObject.transform.GetChild (0).transform.GetComponent <Text>
();//беремо перший дочірній елемент
        if (!PlayerPrefs.HasKey ("Slow Time")) {// чи існує таке значення
            PlayerPrefs.SetInt ("Slow Time", 2);// якщо не існує, тобто вперше запущена гра,
то за замовчуванням у нас 2 безкоштовні уповільнення
            countSlow.text = "2";// виводимо текстове значення на екран
        } else
            countSlow.text = PlayerPrefs.GetInt ("Slow Time").ToString ();//виводимо, що
залишилося

        if (PlayerPrefs.GetInt ("Slow Time") == 0) // закінчилися уповільнення
            GetComponent <Image> ().sprite = inactive;// зміна зображення на неактивну при
перезапуску гри
    }

}
```

```

        else
            GetComponent <Image> ().sprite = active;
    }

    void OnMouseDown () { // відстежувати натискання
        if (PlayerPrefs.GetInt ("Slow Time") > 0 && Time.timeScale != 0.5f) { // якщо їх більше,
            ніж 0 і час ще не уповільнено
                if (PlayerPrefs.GetString ("Music") != "no") // якщо музика не вимкнена
                    GetComponent <AudioSource> ().Play (); // граємо аудіо уповільнення

                PlayerPrefs.SetInt ("Slow Time", PlayerPrefs.GetInt ("Slow Time") - 1); // нинішня
                кількість уповільнень -1
                countSlow.text = PlayerPrefs.GetInt ("Slow Time").ToString (); // та зберігаю нове
                значення
                if (PlayerPrefs.GetInt ("Slow Time") == 0) // додаткова перевірка
                    GetComponent <Image> ().sprite = inactive; // зміна зображення під час
                гри
                StartCoroutine (slowCar ()); // запуск корутини уповільнення часу
            }
        }

        IEnumerator slowCar () { // корутина для уповільнення часу
            Time.timeScale = 0.5f; // уповільнення вдвічі
            yield return new WaitForSeconds (3f); // чекаємо 3 секунди
            Time.timeScale = 1f; // повертаємо звичайний час
        }

        void OnDisable () { // коли переходимо на іншу сцену або оновлюємо сцену, повертаємо
            звичайний час
            Time.timeScale = 1f;
        }
    }
}

```

A.21 FistStepStudy.cs

```

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

[RequireComponent (typeof (MoveCar))]
public class FistStepStudy : MonoBehaviour {

    public GameObject secondCar, carExhaust; // secondCar – змінна, яка відкриває другу машинку
    public Text study; // виведення тексту першого кроку
    private bool firstStepStudy; // змінна служить, щоб показати, що перший етап пройдено

    void Start () {
        GetComponent <MoveCar> ().speedCar = 0f; // спочатку машинка не рухається (бо
        спочатку у нас буде невелике затемнення)
        Invoke ("MoveCar", 1f); // машинка починає їхати трохи пізніше, тобто метод починає
        виконуватись через 1 секунду
    }

    void Update () {
        if (transform.position.x < 16f && !firstStepStudy) { // якщо машинка дійшла до позиції 16
        по x (перший етап ще не виконано)
            firstStepStudy = true; // виконано перший етап
            GetComponent <MoveCar> ().speedCar = 2f; // уповільнюємо машинку
            study.text = "tap the car to accelerate it"; // виводимо першу підказку
        }
    }
}

```

```

void MoveCar () {
    GetComponent <MoveCar> ().speedCar = 12f;// встановили швидкість машинки 12
}

void OnMouseDown () { // прискорення при натисканні на машинку
    if (firstStepStudy) { // якщо машинка їде повільно
        GetComponent <MoveCar> ().speedCar = 30f;// збільшуємо швидкість
        study.text = "";// ховаємо текст

        GameObject ex = Instantiate (carExhaust,
            new Vector3 (gameObject.transform.position.x, 0.2f,
gameObject.transform.position.z),
            Quaternion.Euler (new Vector3 (90, 0, 0))) as GameObject;// створення
вихлопу
        Destroy (ex, 1f);
    }
}

void OnDisable () { // при знищенні першої машинки активується друга
    secondCar.SetActive (true);
}
}

```

A.22 SecondStepStudy.cs

```

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

public class SecondStepStudy : MonoBehaviour {

    public Text study;// виведення тексту другого кроку

    void Start () {
        study.text = "watch where the car turns";// виведення другої підказки
    }

    void OnDisable () { // коли друга машинка знищена
        PlayerPrefs.SetString ("Study", "Done");// відзначаємо, що навчання пройдено
        SceneManager.LoadScene ("game");// завантажуюмо сцену гри
    }
}

```

A.23 BuySlowTime.cs

```

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

public class BuySlowTime : MonoBehaviour {

    public AudioClip success, fail;// змінні для звуку успішної та не успішної покупки
    public Text count, coins;// змінні виведення кількості уповільнень та кількості монеток

    void Start () {
        count.text = PlayerPrefs.GetInt ("Slow Time").ToString ();//задаємо значення змінної при
        старті
    }
    void OnMouseUpAsButton () {

```

```

        if (PlayerPrefs.GetInt ("Coins") >= 100) { // якщо у нас більше або дорівнює 100 монеток,
то ми можемо купувати уповільнення
            PlayerPrefs.SetInt ("Slow Time", PlayerPrefs.GetInt ("Slow Time") + 1); // купуємо
уповільнення
            count.text = PlayerPrefs.GetInt ("Slow Time").ToString (); // нове значення
виводимо в текстовій змінній
            PlayerPrefs.SetInt ("Coins", PlayerPrefs.GetInt ("Coins") - 100); // віднімаємо від
загальної суми 100 монеток
            coins.text = PlayerPrefs.GetInt ("Coins").ToString (); // виводимо нове значення
монеток
            if (PlayerPrefs.GetString ("Music") != "no") { // якщо музика увімкнена
                gameObject.transform.GetChild (0).GetComponent <AudioSource> ().clip = success; // звук при
успішній покупці (звук на дочірньому об'єкті)
                gameObject.transform.GetChild (0).GetComponent <AudioSource> ().Play
                ();
            }
        } else {
            if (PlayerPrefs.GetString ("Music") != "no") { // якщо музика увімкнена
                gameObject.transform.GetChild (0).GetComponent <AudioSource> ().clip = fail; // звук при не
успішній покупці
                gameObject.transform.GetChild (0).GetComponent <AudioSource> ().Play
                (); // відтворити звук
            }
        }
    }
}

```

A.24 BuyNewLocation.cs

```

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

public class BuyNewLocation : MonoBehaviour {

    public AudioSource soundObj; // змінна звуку натискання кнопки покупки
    public AudioClip success, fail; // змінні для звуку успішної та не успішної покупки
    public GameObject checkedBtn; // вибір кнопки
    public Sprite uncheckedBtn, checkedBtnSprite; // спрайт невибраної локації і спрайт вибраної
локації
    public GameObject[] notSelectLocationBtns; // масив кнопок зі зміною зображень активна та
неактивна (кнопка вибору)
    public GameObject[] btnsToBuy; // масив кнопок для покупки локації
    public Text coins; // змінна виведення кількості монеток
    [SerializeField]
    private int locationPrice; // ціна за яку ми купуємо локацію
    [SerializeField]
    private string locationName; // вибір локації при покупці

    void Awake () { // викликається до методу start. У скрипті SelectLocation у методі start
перевірятимемо відповідність сцен з обраними локаціями. Якщо локація відкрита, ми показуємо кнопки вибору.
        if (PlayerPrefs.GetString (locationName) == "Open") { // якщо відкрита якась локація
            for (int i = 0; i < btnsToBuy.Length; i++) { // приховуємо кнопки "купити"
                btnsToBuy[i].SetActive (false); // деактивуємо їх
                checkedBtn.SetActive (true); // показуємо кнопки вибору локації (активна чи неактивна)
            }
        }
    }

    void OnMouseDownAsButton () {

```

```

        if (PlayerPrefs.GetInt ("Coins") >= locationPrice) { // якщо у нас є достатня кількість
монет
            if (PlayerPrefs.GetString ("Music") != "no") { // якщо музика увімкнена
                soundObj.clip = success; // звук при успішній покупці
                soundObj.Play ();
            }
            PlayerPrefs.SetString ("Location", locationName); // виставляємо значення для
локації
            PlayerPrefs.SetString (locationName, "Open"); // відкриваємо локацію
            PlayerPrefs.SetInt ("Coins", PlayerPrefs.GetInt ("Coins") - locationPrice); // покупка
            coins.text = PlayerPrefs.GetInt ("Coins").ToString (); // виведення монеток, що
залишилися
            for (int i = 0; i < btnsToBuy.Length; i++) // кнопки в масиві можемо приховувати
            (кнопки покупок)
                btnsToBuy[i].SetActive (false); // деактивуємо їх

            for (int i = 0; i < notSelectLocationBtns.Length; i++)
                notSelectLocationBtns[i].GetComponent <Image> ().sprite =
uncheckedBtn; // зображення для не вибраних локацій змінюється на неактивне
                checkedBtn.SetActive (true); // активація кнопки
                checkedBtn.transform.GetChild (0).GetComponent <Image> ().sprite =
checkedBtnSprite; // обрана локація змінюється на активну
            } else {
                if (PlayerPrefs.GetString ("Music") != "no") {
                    soundObj.clip = fail; // звук при не успішній покупці
                    soundObj.Play ();
                }
            }
        }
    }
}

```

A.25 SelectLocation.cs

```

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

public class SelectLocation : MonoBehaviour {

    public GameObject[] notCurrentLocationBtns; // для приховування кнопок вибору
    public Sprite thisOne, notThisOne; // зображення кнопок вибору локації
    [SerializeField]
    private string nameOfLocation; // змінна вибору локації

    void Start () {
        if (!PlayerPrefs.HasKey ("Location")) // якщо немає PlayerPrefs зі значенням Location
        (локація не вибрана)
            PlayerPrefs.SetString ("Location", "Suburb"); // то встановлюємо значення за замовчуванням, як
Suburb
        if (PlayerPrefs.GetString ("Location") == nameOfLocation) { // ця локація є нинішньою
            gameObject.transform.GetChild (0).transform.GetComponent <Image> ().sprite =
thisOne; // показую активну кнопку
        } else {
            gameObject.transform.GetChild (0).transform.GetComponent <Image> ().sprite =
notThisOne; // неактивна кнопка
        }
    }

    void OnMouseUpAsButton () { // перехід між локаціями (зміна локацій)

```

```

        PlayerPrefs.SetString ("Location", nameOfLocation); // вибираємо локацію

        for (int i = 0; i < notCurrentLocationBtns.Length; i++)
            notCurrentLocationBtns [i].GetComponent <Image> ().sprite = notThisOne; // зміна
кнопок на неактивні
        gameObject.transform.GetChild (0).GetComponent <Image> ().sprite = thisOne; // зміна
кнопки на активну
    }

}

```

A.26 ShowLocation.cs

```

using System.Collections;
using UnityEngine;

public class ShowLocation : MonoBehaviour {

    public GameObject[] locations; // масив із трьох локацій

    void Start () { // при старті гри відкривається якась локація
        if (PlayerPrefs.HasKey ("Location")) { // якщо локацію вибрано
            for (int i = 0; i < locations.Length; i++) { // проходимо всі локації
                if (PlayerPrefs.GetString ("Location") != locations [i].name) // перевіряємо
на відповідність значень
                    locations [i].SetActive (false); // закриваємо невибрані локації
            }
        } else { // якщо ні, то за замовчуванням вибирається перша локація
            locations [0].SetActive (true); // активуємо потрібну локацію
            for (int i = 1; i < locations.Length; i++) {
                if (PlayerPrefs.GetString ("Location") != locations [i].name)
                    locations [i].SetActive (false); // приховуємо решту
            }
        }

        Camera.main.transform.position = new Vector3 (13.97f, Camera.main.transform.position.y,
Camera.main.transform.position.z); // пересуваємо камеру на певну позицію
    }

}

```

A.27 ExhaustEffectSound.cs

```

using UnityEngine;

public class ExhaustEffectSound : MonoBehaviour {

    public AudioClip[] musicClips; // масив звуків прискорення машинки

    void Start () {
        if (PlayerPrefs.GetString ("Music") != "no") { // якщо музика увімкнена
            GetComponent <AudioSource> ().clip = musicClips [Random.Range (0,
musicClips.Length)]; // випадковий звук прискорення машинок
            GetComponent <AudioSource> ().Play (); // відтворюємо звук
        }
    }

}

```

A.28 ExplodeEffectSound.cs

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

public class ExplodeEffectSound : MonoBehaviour {

    void Start () {
        if (PlayerPrefs.GetString ("Music") != "no")// якщо музика увімкнена
            GetComponent <AudioSource> ().Play ();//програємо звук при зіткненні
    }

}
```

A.29 CreatingAudio.cs

```
using UnityEngine;

public class CreatingAudio : MonoBehaviour {

    public GameObject music;// змінна music

    void Start () {
        if (GameObject.Find ("Audio Control(Clone)") == null)//перевіряю чи створено об'єкт(не
створюю повторно)
            Instantiate (music, new Vector3 (0,0,0), Quaternion.identity);// звук не (зникає)
знищується під час переходу на інші сцени. Створюю об'єкт.
    }

}
```

A.30 AudioControl.cs

```
using System.Collections;
using UnityEngine;

public class AudioControl : MonoBehaviour {

    private bool play;// змінна показує, чи грає музика

    void Start () {
        DontDestroyOnLoad (transform.gameObject);// об'єкт не буде знищуватись при
завантаженні нової сцени
        StartCoroutine (carHorn ());//корутина сигналу машини
    }

    void Update () {// вмикаємо або вимикаємо музику залежно від того, чи натиснуто кнопку
        if (PlayerPrefs.GetString ("Music") != "no" && !play) {// якщо музика увімкнена і не грає
            GetComponent <AudioSource> ().Play ();//вмикаємо музику
            play = true;
        } else if (PlayerPrefs.GetString ("Music") == "no" && play) {// якщо музика вимкнена та
грає
            GetComponent <AudioSource> ().Pause ();//ставимо на паузу
            play = false;
        }
    }

    IEnumerator carHorn () {// сигнал машинки
        while (true) {// постійно повторюються
```

```
yield return new WaitForSeconds (Random.Range (12f, 22f));// чекаємо кілька  
секунд  
        if (PlayerPrefs.GetString ("Music") != "no")// якщо музика увімкнена  
            transform.GetChild (0).GetComponent <AudioSource> ().Play ();//програємо звук  
    }  
}
```

ДОДАТОК Б
Слайди презентації

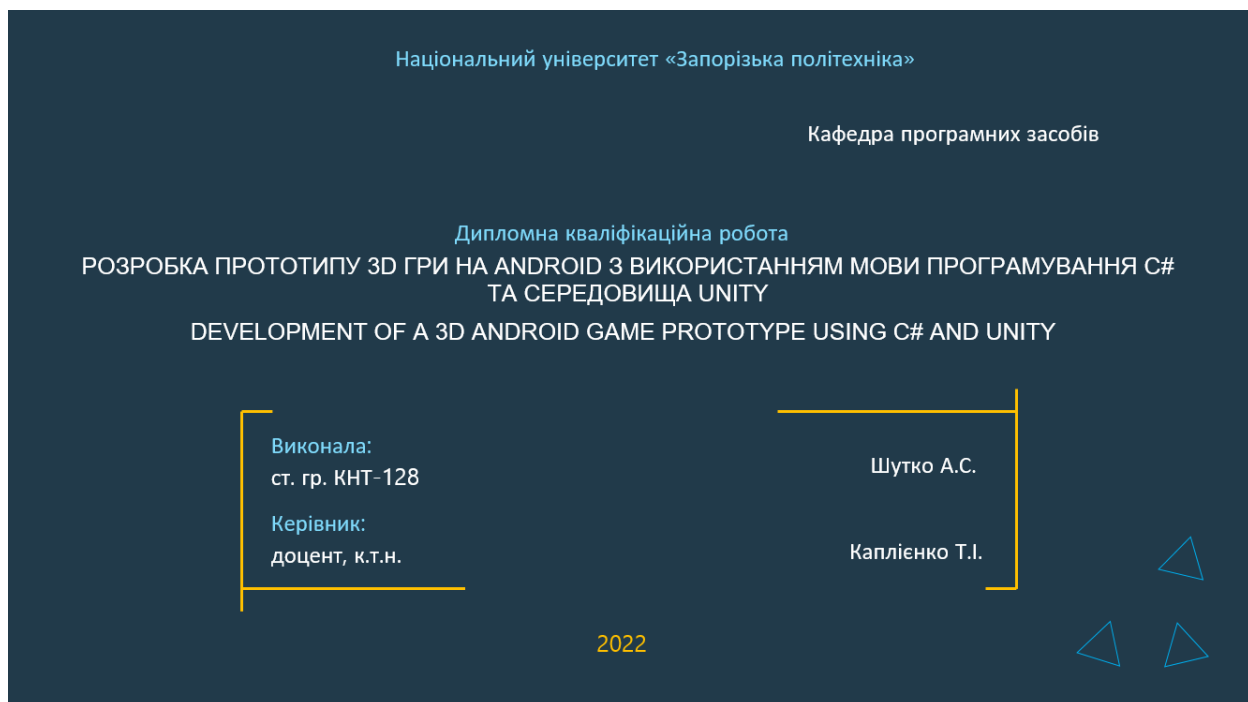


Рисунок Б.1 – Слайд №1

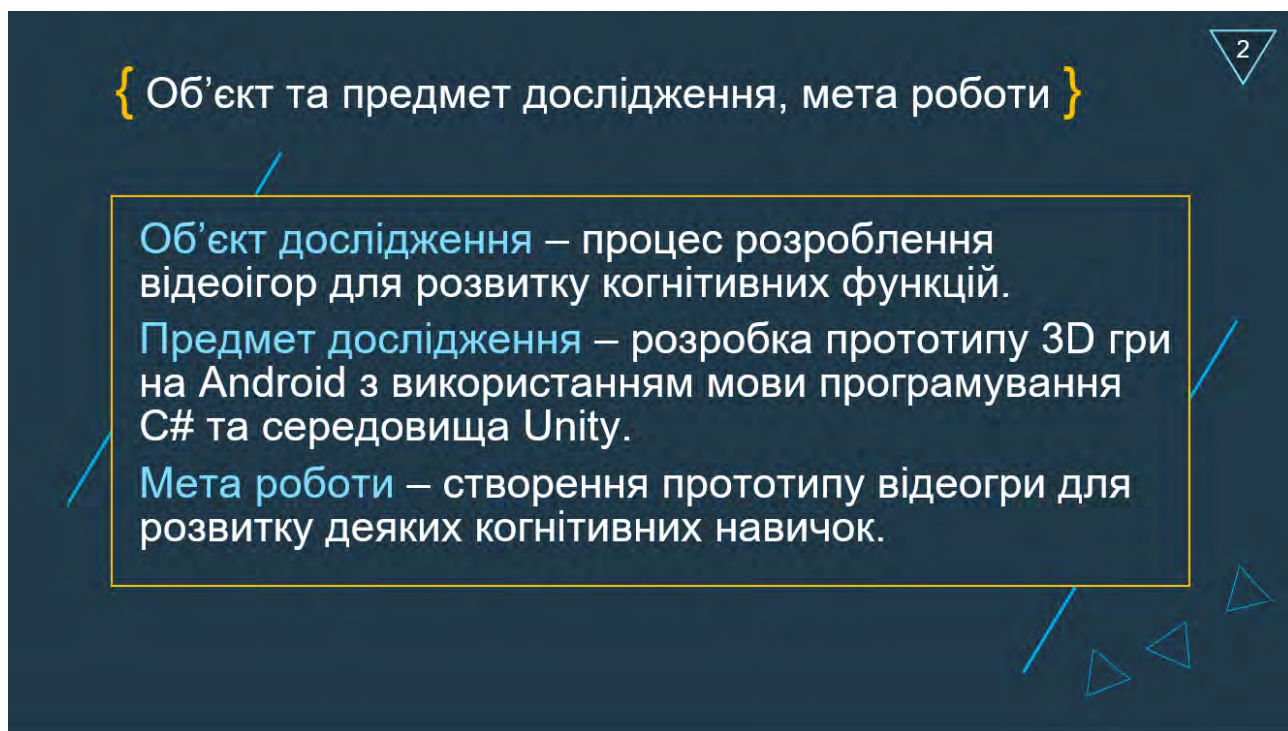


Рисунок Б.2 – Слайд №2

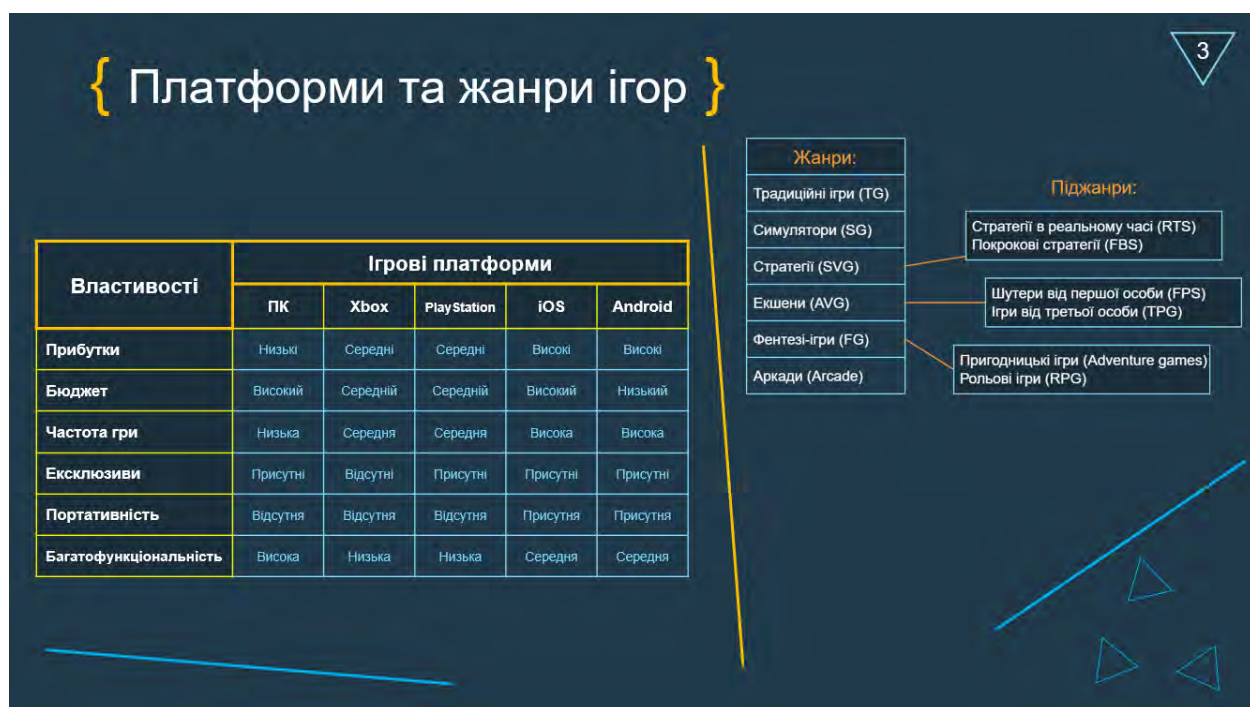


Рисунок Б.3 – Слайд №3



Рисунок Б.4 – Слайд №4

{ Вибір ігрового рушія }

5

Параметри порівняння	Ігровий рушій	
	Unity	Unreal Engine
Платформи, що підтримуються	Великий вибір	Великий вибір
Графіка	Висока. Але більше роботи із графікою.	Відмінна
Відкритий вихідний код	Тільки для тих, хто купив платну версію.	Для всіх абсолютно безкоштовний та доступний на гітхабі.
Кількість навчальних матеріалів	Багато	Мало
Кількість ассетів (моделі, анімації, шейдери тощо)	Величезний вибір платних та безкоштовних ассетів	Менший вибір платних та безкоштовних ассетів
Для кого рушій	Для невеликої команди або одного розробника	Для великої команди

Рисунок Б.5 – Слайд №5

{ Вибір середовища розробки та мови програмування }

6

Властивості	Середовища розробки	
	Monodevelop	Visual Studio
Продуктивність	Добре	Є проблеми
Стабільність	Найменш стабільне	Стабільне
Плагіни	Немає	Так
Стилізація коду	Немає	Так
Підтримка спільноти	Має меншу підтримку спільноти	Має велику підтримку спільноти

Властивості	Мови програмування	
	C#	UnityScript
Швидкість	Висока	Висока
Читабельність	Відмінна	Добра
Сумісність з VS або MonoDevelop	Так	Немає
К-ть бібліотек у вільному доступі	Багато	Багато
Підтримка спільноти	Так	Немає
Строго типізований	Так	Немає

Рисунок Б.6 – Слайд №6

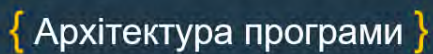


Рисунок Б.7 – Слайд №7

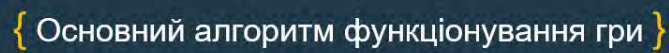


Рисунок Б.8 – Слайд №8

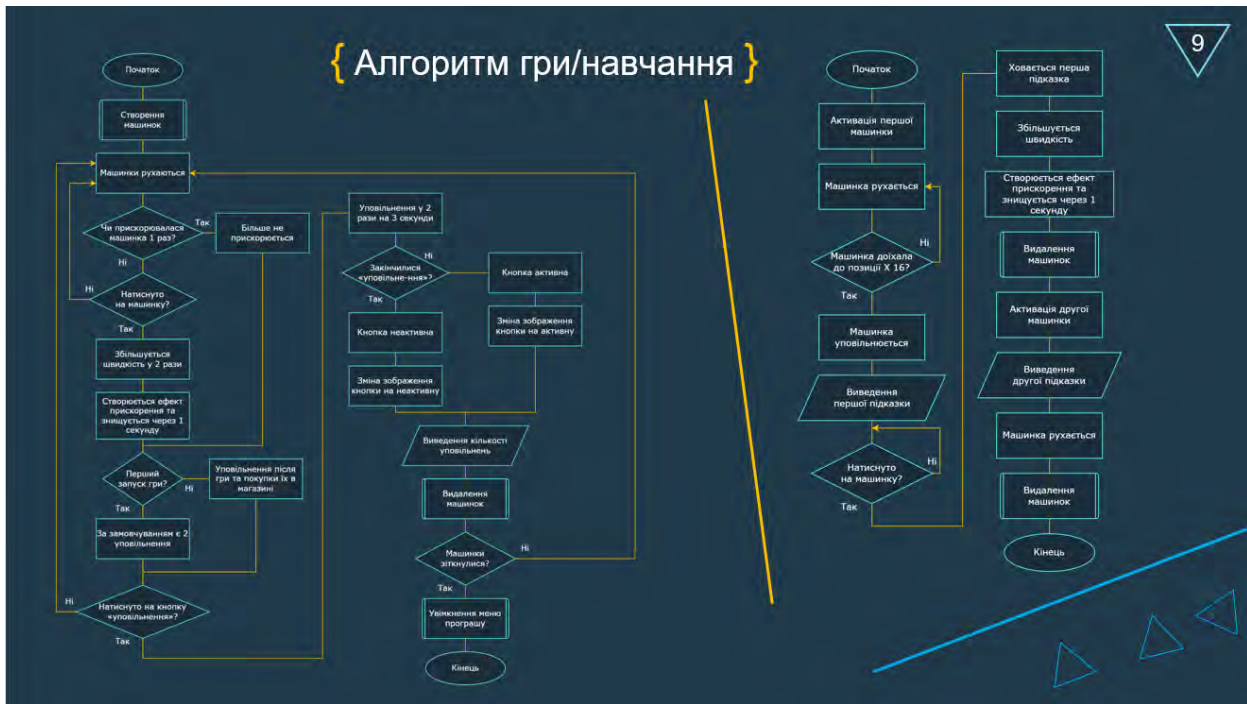


Рисунок Б.9 – Слайд №9



Рисунок Б.10 – Слайд №10



Рисунок Б.11 – Слайд №11

12

{ Класи розробленої гри }

Клас	Опис
ButtonsPressing	Скрипт, що містить у собі функціонал кнопок
FadingScreen	Скрипт ефекту згасання сцени
MoveCar	Скрипт, що контролює рух машинок
WestTurnLeft	Скрипт західного повороту машини ліворуч
WestTurnRight	Скрипт західного повороту машини праворуч
EastTurnLeft	Скрипт східного повороту машини ліворуч
EastTurnRight	Скрипт східного повороту машини праворуч
NorthTurnLeft	Скрипт північного повороту машини ліворуч
NorthTurnRight	Скрипт північного повороту машини праворуч
SouthTurnLeft	Скрипт південного повороту машини ліворуч
SouthTurnRight	Скрипт південного повороту машини праворуч
CreatingCars	Скрипт створення машинок
DeleteCar	Скрипт видалення машинок
Acceleration	Скрипт прискорення машинок
CollisionCar	Скрипт, який відстежує, якщо машинки зіткнулися, то гра завершується

Клас	Опис
CarLights	Скрипт відповідає за ввімкнення та вимкнення будь-якого поворотника машинок
ScoreShow	Скрипт, що показує рахунок гравця
ShowCoins	Скрипт, показує монети гравця
ShowLoseGameMenu	Скрипт виведення меню програшу
SlowTime	Скрипт уповільнення машини
FirstStep	Скрипт першого кроку навчання у грі
SecondStepStudy	Скрипт другого кроку навчання у грі
BuySlowTime	Скрипт придбання уповільнення в магазині
BuyNewLocation	Скрипт придбання нової локації у магазині
SelectLocation	Скрипт для вибору локації гри
ShowLocation	Скрипт для відображення вибраної локації гри
ExhaustEffectSound	Скрипт звуку ефекту прискорення машини під час натискання на неї
ExplodeEffectSound	Скрипт звуку ефекту зіткнення машинок, тобто вибуху
CreatingAudio	Скрипт, який відповідає за зникнення звуку під час переходу між сценами
AudioControl	Скрипт контролю аудіо (Звуки самої гри та гудків машин)

Рисунок Б.12 – Слайд №12



Рисунок Б.13 – Слайд №13

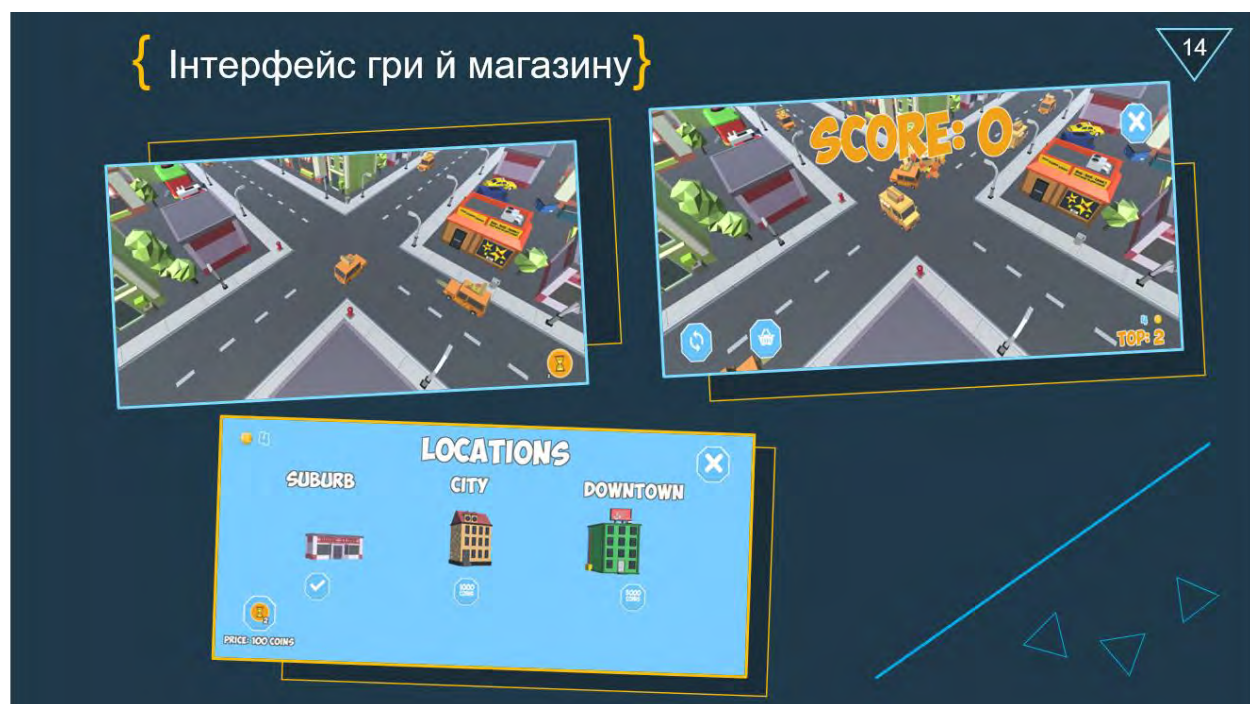


Рисунок Б.14 – Слайд №14

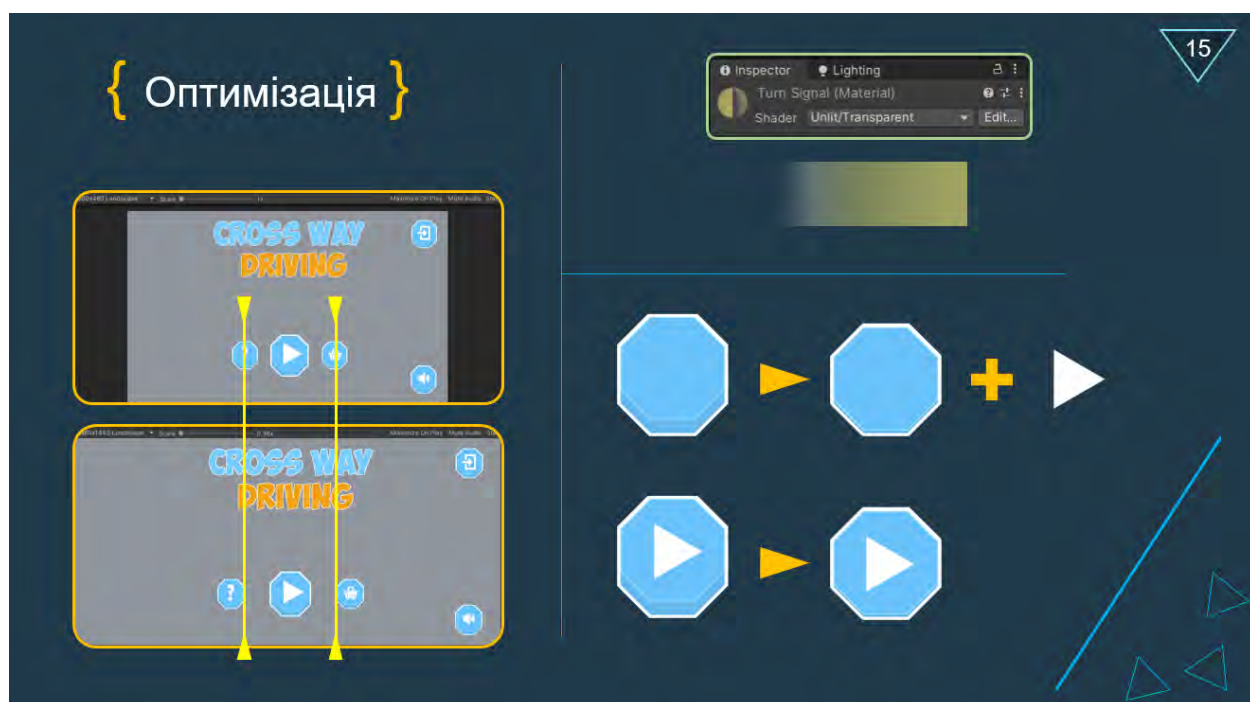


Рисунок Б.15 – Слайд №15

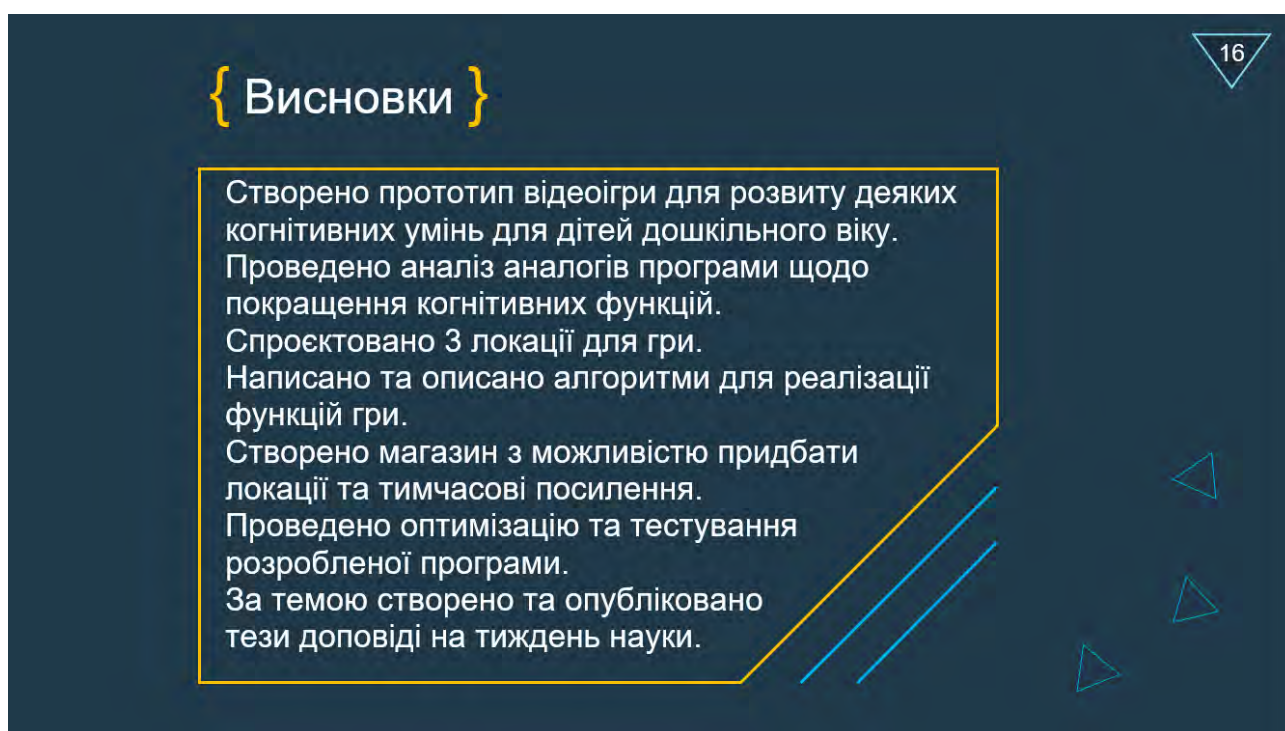


Рисунок Б.16 – Слайд №16

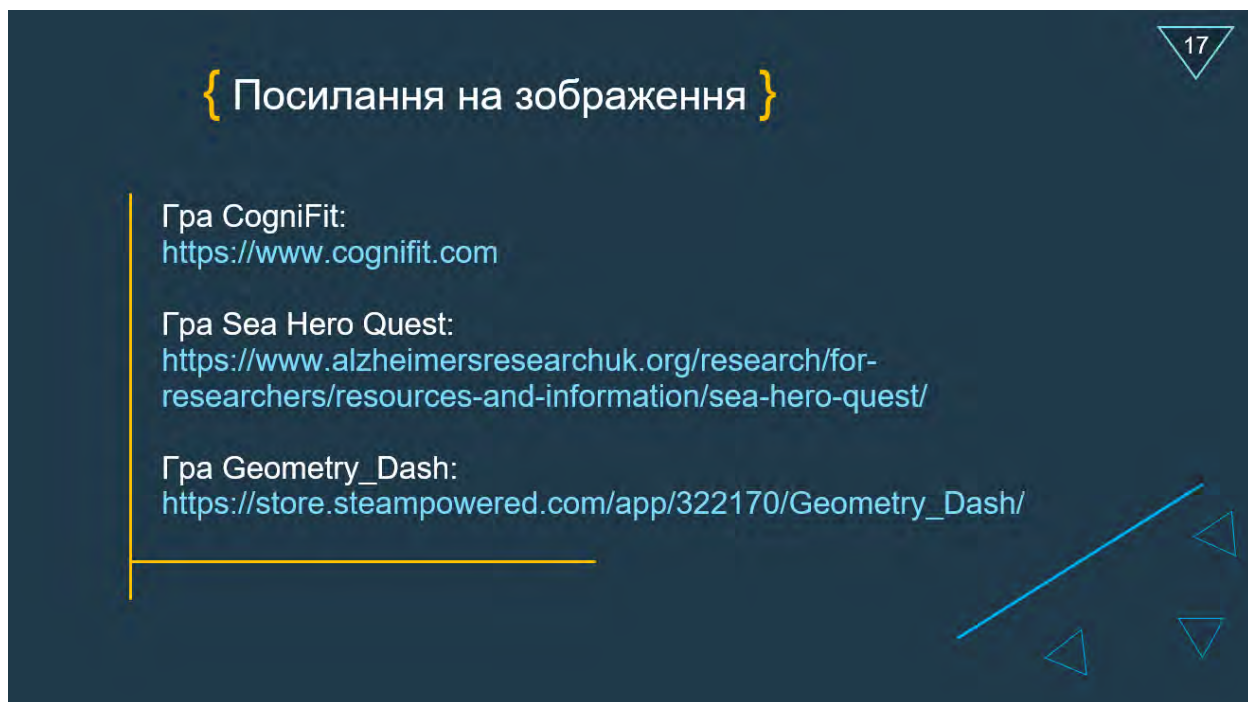


Рисунок Б.17 – Слайд №17