

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук та технологій
(повне найменування факультету)

Кафедра комп'ютерних систем та мереж
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавра

(ступінь вищої освіти (освітній ступінь))

на тему **РОЗРОБКА АРКАДНОЇ СПОРТИВНОЇ 2D ГРИ «ICE BLITZ» НА БАЗІ
ФРЕЙМВОРКУ UNITY**

Виконав: студент 4 курсу, групи КНТЗ-511сп
спеціальності _____

123 «Комп'ютерна інженерія»

(код і найменування спеціальності)

Освітня програма (спеціалізація) _____

«Комп'ютерна інженерія»

КУНТЯК А. С.

(ПРИЗВИЩЕ та ініціали)

Керівник КУДЕРМЕТОВ Р.К.

(ПРИЗВИЩЕ та ініціали)

Рецензент СТЕПАНЕНКО О.О.

(ПРИЗВИЩЕ та ініціали)

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

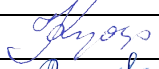
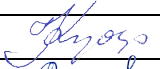
Факультет Комп'ютерних наук і технологій
 Кафедра «Комп'ютерні системи та мережі»
 Ступінь вищої освіти бакалаврський
 Спеціальність 123 Комп'ютерна інженерія
 (код і найменування)
 Освітня програма (спеціалізація) Комп'ютерна інженерія
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
 Завідувач кафедри КУДЕРМЕТОВ Р.К.
 « » 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

- КУТНЯКА Андрія Сергійовича
 (ПРИЗВИЩЕ, ім'я, по батькові)
1. Тема проекту (роботи) Розробка аркадної спортивної 2D гри «Ice Blitz» на базі фреймворку Unity
 керівник проекту (роботи) к. т. н., доцент, КУДЕРМЕТОВ Равіль Камілович
 (науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)
- затверджені наказом вищого навчального закладу від «01» квітня 2024 року № 105
2. Строк подання студентом проекту (роботи) 26 травня 2024 року
3. Вихідні дані до проекту (роботи) предметна область, види матеріальних об'єктів, види інформаційних об'єктів, комплекс технічних засобів, стандарти або інші обмежувальні документи, технології та інструменти для проектування тощо
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Загальне уявлення про розробку ігор
Розробка гри
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Діаграма прецедентів
Сайт гри «Ice Blitz»
Діаграми варіантів послідовності

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-2	КУДЕРМЕТОВ Р. К., к.т.н., доцент		
нормоконтроль	ПОЛЬСЬКА О.В., ст. викладач		

7. Дата видачі завдання 18.03.2024 р.

КАЛЕНДАРНИЙ ПЛАН

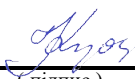
№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Надання та затвердження «Технічного завдання» на ДРБ	до 15.04	
2	Аналіз технічного завдання та робота з джерелами інформації	16.04 - 17.04	
3	Опис предметної області.	18.04 – 20.04	
4	Складання дизайн-документа	22.04 – 24.04	
5	Підбір готових ассетів	25.04 – 27.04	
6	Створення проєкту	29.04 – 04.05	
7	Оформлення ПЗ ДРБ та графічних матеріалів	06.05 - 26.05	
8	Проходження нормоконтролю	27.05 - 29.05	
9	Перевірка на наявність академічного плагіату	30.05 - 01.06	
10	Рецензування	02.06 - 05.06	
11	Захист ДРБ	06.06	

Студент (ка)


(підпис)

Андрій КУТНЯК
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)


(підпис)

Равіль КУДЕРМЕТОВ
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

ПЗ. 76 с., 15 рис., 11 ліст., 15 джерел.

КОМП'ЮТЕРНІ ІГРИ, UNITY, ФРЕЙМВОРК, WebGL, БРАУЗЕРНА ГРА, C#

Об'єкт розробки – комп'ютерна гра на Unity для браузерів з підтримкою технології WebGL.

Мета роботи – розробка додатку розважального характеру, розрахований для браузерів, які підтримують технологію WebGL.

У першій частині було проведено детальний огляд у сфері розробки ігор, на які ролі розподіляються команди розробників, їх обов'язки, а також етапи виробництва. Також були розглянуті що таке ігровий двигун (або фреймворк) та які їх особливості на прикладі Unity, на якій розроблена гра.

У другій частині роботи більш детально розглянуті етапи створення програми для розваги, а саме: складаний дизайн-документ, створені візуальні та звукові елементи, розроблений ігровий процес з урахуванням можливостей фреймворку, використані скрипти на мові програмування C#. Також проведені тестування, опублікування на спеціалізованому ігровому порталі для незалежних розробників «itch.io» [16] та передбачена подальша підтримка готового варіанту.

ЗМІСТ

Вступ.....	7
1 Загальне уявлення про розробку ігор.....	9
1.1 Ролі.....	9
1.1.1 Продюсер.	9
1.1.2 Видавець.	10
1.1.3 Команда розробки.	11
1.2 Документування проекту.	14
1.3 Процес розробки.....	15
1.4 Передпродакшн.	15
1.5 Виробництво.....	17
1.6 Підтримка.	18
1.7 Аутсорсинг.....	18
1.8 Ігровий движок (фреймворк).	19
1.9 Особливості ігрового движка Unity	20
2 Розробка гри	23
2.1 Написання дизайн-документа.....	23
2.1.1 Шаблон десятисторінника від Скота Роджерса.....	23
2.1.2 Дизайн-документ гри «Ice Blitz»	25
2.2 Створення візуальної частини	29
2.3 Написання музики та звукових ефектів	32
2.4 Складання ігрового процесу	33
2.4.1 Діаграми варіантів використання	33
2.4.2 Діаграми послідовності.....	35
2.4.3 Особливості WebGL в UNITY	37
2.4.4 Створення інтерфейсу	39
2.4.5 Як проектувати інтерфейс користувача	43
2.4.6 Написання скриптів.....	45

2.5 Тестування	70
2.6 Реліз.....	72
2.7 Післявипускний цикл гри: забезпечення постійного успіху та задоволення гравців.....	73
Висновки.....	75
Перелік джерел посилання	76

ВСТУП

Розробка комп'ютерних ігор є складним та багатогранним процесом, що включає в себе різні етапи від початкового планування до фінального релізу продукту. Сучасні технології, дозволяють створювати високоякісні розважальні програми, які можуть працювати в безлічі ігрових платформ.

У цій роботі розглянуто ключові аспекти розробки гри, зосереджуючись на технічних та творчих елементах процесу.

Об'єктом дослідження є комп'ютерна гра для браузерів з підтримкою технології WebGL.

Предметом дослідження є засоби створення комп'ютерних ігор та етапи їх розробки.

Метою роботи є розробка гри, розрахований для браузерів, які підтримують WebGL, використовуючи фреймворк Unity.

Створення розважальної програми складається з наступних етапів:

- складання дизайн-документа, яка містить інформацію по те, якою гра буде розроблена. Важливо зазначити, що під час розробки з різноманітних причин можуть бути відхилення від документації, яка була вже створена;

- створення або придбання необхідних візуальних та звукових елементів, такі як спрайти або 3D-моделі, звукові ефекти та музику;

- розробка ігрового процесу з використанням можливостей Unity. Даний фреймворк відомий своєю гнучкістю та доступністю у розробці ігор. Завдяки інтеграції з WebGL, розробники можуть створювати ігри, які запускаються у будь-якому сучасному веб-браузері без необхідності встановлення додаткових плагінів або програмного забезпечення;

- протестувати наявність багів з подальшими їх усуненнями;

- випустити готову програму заховив на сайті.

Деякі з цих етапів розробки можуть виконуватися як паралельно, так й послідовно в залежності від розміру команди розробників та завчасно побудованого плану виробництва.

Основні результати:

- складаний дизайн-документ гри «Ice Blitz», аркадної спортивної 2D гри, натхненниками для якої були ігри на 8-бітних приставках від компанії Nintendo під навами «Ice Hockey» та «Blades of Steel». Даний документ містить короткий опис сюжету, команди гравців, візуальний та звуковий стиль, у якій буде розроблена гра, опис ігрового процесу, інтерфейсу та ігрових механіків, правила або набору правил взаємодії гравця з грою;

- підібрані та придбані усі необхідні візуальні та звукові елементи, включаючи музику, звукові ефекти та спрайти, які є за посиланнями [12-15];

- розроблений прототип гри з використанням можливостей, які надає Unity;

- випущена браузерна гра на сайті для незалежних розробників «itch.io».

Браузерні ігри не потребують встановлення додаткових програм або додаткових витрат часу на завантаження. Достатньо відкрити веб-браузер, зайти на сторінку сайту і можна грати.

1 ЗАГАЛЬНЕ УЯВЛЕННЯ ПРО РОЗРОБКУ ІГОР

Комп'ютерні ігри може розробляти як одна людина, так й команда розробників (компанія). Команди розробників можуть спеціалізуватися на створенні комп'ютерних, консольних або мобільних ігор.

Розробка може фінансуватися іншою великою компанією, яка називається видавцем (компанія, яка розповсюджує ігри, розроблені компаніями-розробниками).

Існує багато різних способів незалежного розповсюдження копій ігор, з участю видавця або без нього, через цифрові платформи-дистриб'ютори та фізичні носії.

Витрати на розробку найбільших ігор оцінюються мільйонами доларів, і протягом останніх двох десятиліть ці бюджети продовжували зростати, так само як і кількість команд розробників, персонал і час розробки.

1.1 Ролі

На початку 1980-х років, коли були випущені перші домашні комп'ютери та ігрові приставки, один програміст міг впоратися з більшістю пов'язаними з розробкою ігор завданнями,. Сучасна розробка ігор передбачає наявність широкого кола фахівців та допоміжного персоналу.

1.1.1 Продюсер

Розробку гри контролюють внутрішні та зовнішні продюсери.

Зовнішні продюсери - це люди, призначені видавцем до команди розробників для управління процесом розробки та контролю бюджету.

Внутрішні продюсери - це люди, які працюють у студії, де розробляється гра.

Вони керують командою розробників, створюють графіки, готують звіти про прогрес і відповідають за наймання та розподіл персоналу. До обов'язків продюсера входять зв'язки з громадськістю, переговори, контракти, робота з персоналом та зацікавленими сторонами, управління розкладом і бюджетом, забезпечення якості, бета-тестування та локалізація. Людей на цій посаді іноді називають проектними менеджерами, керівниками проектів або директорами.

1.1.2 Видавець

Видавці комп'ютерних ігор - це компанії, які публікують і випускають комп'ютерні ігри. Ігри можуть бути розроблені видавцем або іншими компаніями. Як і видавці книг та DVD-фільмів, видавці комп'ютерних ігор відповідають за виробництво та маркетинг своєї продукції, включаючи дослідження ринку та рекламу.

Видавці зазвичай фінансують розробку, яка оплачується компаніям-розробникам комп'ютерних ігор (видавці називають це аутсорсинговою розробкою) або власним співробітникам у студіях. Як правило, видавець володіє правами на гру. Великі видавці комп'ютерних ігор також займаються розповсюдженням ігор, які вони видають, а деякі менші видавці передають ігри, які вони видають, дистриб'юторським компаніям (або великим видавцям комп'ютерних ігор).

Інші функції, які зазвичай виконують видавці, включають оплату ліцензій, використаних у грі (наприклад, ліцензії на музику та торгові марки), оплату локалізації та створення елементів графічного дизайну, таких як дизайн коробки та обкладинки гри.

Великі видавці також можуть надавати такі послуги, як звукове оформлення та набори кодів для загальних функцій, що може підвищити ефективність роботи всієї внутрішньої та зовнішньої команди розробників.

Видавці намагаються управляти ризиками розробки, надаючи фінансування на розробку ігор продюсерам і менеджерам проектів. Це дозволяє

видавцям контролювати процес розробки та надавати допомогу в разі потреби. Більшість комп'ютерних ігор, розроблених сторонніми розробниками, отримують регулярні аванси у вигляді роялті. Ці аванси виплачуються, коли розробник досягає певного етапу розробки, відомого як віха.

Незалежні розробники можуть випускати свої ігри без видавця і самостійно обирати методи розповсюдження, а можуть звернутися за допомогою до видавця для просування і розповсюдження своїх ігор.

1.1.3 Команда розробки

Команди розробників у компаніях варіюються від невеликих команд з двох-трьох людей, які створюють казуальні ігри, до великих команд із сотень і тисяч працівників, які створюють масштабні ігри. Усередині компаній розподіляються підзадачі, пов'язані з розробкою ігор. Окремі посади можуть бути різними, але їхня залученість в індустрію однакова. Існують такі випадки, коли декілька працівників виконують одну роль, тоді як один і той самий працівник може виконувати декілька з них. Найбільша кількість людей у команді - художники, за ними йдуть програмісти, гейм-дизайнери, звукорежисери і два-три продюсери, відповідальні за менеджмент. Люди на цих посадах працюють повний робочий день. Інші професії, наприклад, тестувальники, можуть працювати неповний робочий день. Зарплата на цих посадах залежить від досвіду та посади працівника.

Команди розробників включають такі посади як гейм-дизайнер, дизайнер рівнів, художник, ігровий програміст, звукорежисер та тестувальник.

Гейм-дизайнер - це людина, яка розробляє правила та структуру гри, а також проектує ігровий процес. У команді розробників зазвичай є провідний гейм-дизайнер, який координує роботу інших гейм-дизайнерів. Вони найкраще розуміють суть гри. Частина роботи гейм-дизайнера полягає в тому, щоб продумати занурюваність у сюжетну частину гри, включаючи діалоги, коментарі, сцени, предмети та підказки. У великих проектах окремі гейм-дизайнери часто відповідають за різні частини гри, такі як механіка гри, набір правил взаємодії гравця з грою, користувацький інтерфейс, персонажі та діалоги.

Дизайнер рівнів - це людина, яка використовує інструменти розробки ігор або інше програмне забезпечення для створення рівнів, завдань або місій для відеоігор. Зазвичай це комерційні програми для 3D чи 2D дизайну або редактори рівнів, розроблені спеціально для конкретної гри.

Дизайнери ігрових рівнів працюють як над незавершеними, так і над завершеними версіями гри. Ігрові програмісти часто розробляють редактори рівнів та інструменти для розробки ігор, які використовують гейм-дизайнери. Це означає, що гейм-дизайнеру не потрібно отримувати доступ до коду гри або вносити зміни в нього. Розробники ігор можуть використовувати мови програмування високого рівня та скриптові мови для визначення взаємодії гравця та ШІ (штучного інтелекту). На відміну від інструментів редагування рівнів, які іноді доступні для ігрової спільноти, гейм-дизайнери часто використовують макети або прототипи, щоб забезпечити узгодженість і чіткість макета.

Художники малюють візуальну частину гри. Художній відділ зазвичай має директора або супервайзера, який також відповідає за те, щоб бачення гри був узгоджений серед колег. Директор очолює відділ, планує та координує діяльність усієї команди розробників.

2D-художники створюють концепт-арт, спрайти, текстури, фонові зображення та зображення місцевості, а також користувацькі інтерфейси, тоді як 3D-художники створюють моделі, полігональні сітки, анімацію, 3D-середовища та кінематику. Художники можуть виконувати обидві ролі (2D і 3D).

Ігрові програмісти - це інженери-програмісти, які здебільшого розробляють комп'ютерні ігри та супутнє програмне забезпечення (наприклад, інструменти для розробки ігор). Програмісти виконують усю роботу, пов'язану з вихідним кодом. Зазвичай є один або кілька провідних програмістів, які реалізують початкову кодову базу, планують подальший розвиток проекту та координують роботу інших програмістів.

Ігрові програмісти можуть бути залучені до розробки наступних елементів:

- фізика - програмування фреймворку, який включає моделювання "фізики" (фізичних моделей), таких як рух об'єктів і зіткнення;

- штучний інтелект (ШІ) - створення комп'ютерних агентів з використанням технології ШІ. Сюди входить написання сценаріїв, планування та логічне програмування;

- графіка - управління графічним контентом і використанням комп'ютерної пам'яті. Розробка графічних рушіїв, інтеграція 3D-моделей і текстур, координація з фізикою;

- звук - інтеграція музики, мови та звукових ефектів за потреби;

- ігровий процес - реалізація різноманітних ігрових механік та функцій;

- кодування - розробка та підтримка високорівневих систем кодування для різних ігрових завдань, таких як ШІ та тригери редактора рівнів;

- інтерфейс користувача - програмування пунктів меню, візуального інтерфейсу, систем допомоги та зворотного зв'язку;

- пристрої введення - експлуатація та налаштування різних пристроїв введення, таких як клавіатура, миша та геймпад;

- мережеві технології - керування доступом до локальних мереж та Інтернету;

- інструменти для розробки ігор - створення додаткового спеціалізованого програмного забезпечення для підтримки розробки ігор. Часто використовується для допомоги дизайнерам і сценаристам.

Звукорежисери - це технічні фахівці, відповідальні за загальний звук і звукове оформлення гри. Вони також відповідають за озвучення та інші звукові ефекти. Композитори, які створюють музику для ігор, також є частиною команди звукорежисерів, але зазвичай їх залучають на аутсорсингу. Творці ігор можуть замовляти музику у музичних продюсерів.

Забезпечення якості здійснюється за допомогою тестування. Тестувальники аналізують комп'ютерні ігри та фіксують помилки, які вони знаходять. Тестування зазвичай вимагає хорошої комп'ютерної грамотності та аналітичних навичок.

Тестувальники перевіряють, чи відповідає гра вимогам дизайну. Іншими словами, вони перевіряють, чи працює гра так, як очікувалося, і чи підходить вона

для гравця. Це включає перевірку всіх функцій, сумісності, локалізації тощо. Тестування необхідне протягом усього процесу розробки, але може бути дорогим і зазвичай активно використовується лише після завершення проекту.

1.2 Документування проекту

З моменту, коли вся команда розробників починає працювати над грою, виникає необхідність документувати процес розробки. З одного боку, це полегшує пошук колег та інвестиційних партнерів, з іншого - допомагає планувати бюджет і координувати розробку. Зміст і перелік документів сильно варіюється залежно від рівня розробника, але можна виділити три основні типи:

- концепт-документ - від двох до шести сторінок тексту, бажано з ілюстраціями, що демонструють основні особливості гри;

- дизайн-документ (або диздок): коли стає зрозуміло, що гра пройде стадію виробництва, до концептуального документу додається більше деталей, і він перетворюється на дизайн-документ;

- документ-пропозиція - короткий опис гри, не включаючи внутрішні деталі розробки, що пояснює потенційним інвесторам, чому гра буде прибутковою;

- технічний документ проекту - у випадку невеликих російських розробників зазвичай не виділяється окремо, але містить технічні вимоги до гри (наприклад, бюджет полігонів, частота кадрів в секунду (fps), обсяг пам'яті, портативність, використання бази даних, анти-чит, багатопоточність) та опис інструментів і мов програмування, які будуть використовуватися.

1.3 Процес розробки

Процес розробки гри зазвичай включає такі етапи:

- підготовка («препродакшн»);
- уточнення геймдизайну;
- виробництво;
- підтримка.

Етапи можуть змінюватися в залежності від переваг фірми та особливостей проекту.

1.4 Передпродакшн

Після створення початкової ідеї гри необхідно оцінити виробничі ризики та бюджет, а також вирішити гуманітарні питання.

Розраховується кількість необхідного контенту та оцінюється час з витратами. Визначається кількість художників, які будуть найняті для гри, їх рівень кваліфікації та термін роботи. Обирається середовище розробки гри та оцінюється рівень графіки. Прототипування основних елементів гри.

Бюджет полігонів - кількість полігонів, яким витрачається на ігрові активи, кількість активів, які можуть співіснувати в кадрі і кількість полігонів, залишається для оточення. Оцінюється потенційна продуктивність сервера для багатокористувацької гри. Визначається бюджет.

Якщо розробник і видавець є окремими компаніями, зазвичай ідеї подаються на затвердження керівництву, а потім на розгляд видавцю. Робоча демонстрація корисна, але не обов'язкова. Після того, як зацікавлений видавець знайдений, можна починати виробництво. У наш час ідея гри рідко буває переконливою, якщо видавець виявляє зацікавленість.

Якщо розробник також є видавцем або частиною того ж відділу в тій же компанії, ідею має схвалити лише топ-менеджмент. Однак, залежно від розміру видавничої компанії, може знадобитися кілька спроб, перш ніж ідея буде прийнята на всіх рівнях управління.

Представником проекту зазвичай є гейм-дизайнер, але це також може бути хтось з ігрової індустрії в іншій якості. Перед початком виробництва гейм-дизайнер повинен написати дизайн-документ (детальний документ з описом концепції та ігрового процесу). Він також може включати попередні ескізи (малюнки, фотографії) різних аспектів гри. Деякі гейм-дизайнери можуть додавати до дизайн-документа грубий прототип, що демонструє один або кілька аспектів гри. Як правило, дизайн-документ - це сукупність усіх або більшої частини матеріалів, пов'язаних з початковою концепцією. Найважливішою характеристикою дизайн-документа є те, що він є "живим"; фактично, остаточний дизайн не є завершеним, поки гра не буде розроблена. Він може змінюватися щотижня, а в деяких випадках і щодня. Тому, хоча проектна документація повинна існувати в тій чи іншій формі до початку повномасштабного виробництва, повна гра рідко є завершеним проектом, навіть якщо вона описує багато аспектів усіх етапів розробки.

Основна команда програмістів і художників може почати роботу над ідеєю ще до того, як буде затверджено дизайн. Програмісти можуть розробити ранній прототип, який демонструє одну або кілька функцій, необхідних для гри. Або ж вони можуть почати розробляти каркас, який згодом буде використаний у грі. Художники можуть малювати ескізи, які стануть основою для розробки власне ігрових ресурсів. Продюсери можуть спочатку працювати над грою неповний робочий день, але в міру розвитку гри можуть збільшувати свою зайнятість.

1.5 Виробництво

Основні етапи виробництва передбачають величезний обсяг роботи. Програмісти пишуть вихідний код, художники малюють графіку (спрайти та 3D-моделі елементів гри). Звукорежисери розробляють звукові ефекти, а композитори пишуть музику для гри. Дизайнери створюють рівні, а сценаристи пишуть сценарії для позаігрового пересування персонажів і діалогів.

Весь цей час гейм-дизайнери доповнюють і змінюють дизайн гри, щоб відобразити своє поточне бачення гри. Іноді функції або рівні видаляються, іноді додаються. Можуть бути випадки, коли покращується художній підхід, або змінюється сюжет. Можуть з'явитися нові цільові платформи або цільові аудиторії. Всі ці зміни спочатку повинні бути задокументовані в проектній документації.

З точки зору часу, перші рівні гри займають найдовше часу. Для створення рівнів дизайнери та художники використовують інструменти, які потребують внутрішніх навичок та модифікацій. Ранні рівні гри можуть зазнавати різних змін, оскільки деякі рівні можуть застарівати з появою нових функцій. Крім того, через динамічну природу розробки ігор, бачення дизайну ранніх рівнів може змінюватися з часом. У типовому трирічному процесі розробки гри перший рівень часто займає місяці. Наступні рівні можна розробляти швидше, оскільки список функцій стає повнішим, а бачення гри - чіткішим.

Тестувальники приєднуються до гри, як тільки вона стає доступною для випробувань. Це може бути один рівень або частина гри, яка може бути використана в межах розумного діапазону. На ранніх стадіях тестування один тестувальник витрачає на гру лише частину часу. Коли гра наближається до завершення, час тестувальника присвячується одній грі, намагаючись протестувати нові функції, які можуть бути регресійно протестовані і які можуть вимагати понаднормової роботи. Тестування дуже важливе, оскільки більшість сучасних ігор є складними, і навіть одна зміна може призвести до катастрофи.

1.6 Підтримка

Під підтримкою зазвичай розуміють випуск патчів для виправлення помилок, виявлених після випуску гри. Однак, наприклад, у випадку масових багатокористувацьких онлайн-ігор (ММО) підтримка може конкурувати або навіть перевершувати виробництво за часом і зусиллями, оскільки успішні ММО вимагають постійного розвитку і розширення, щоб запобігти відтоку гравців.

1.7 Аутсорсинг

Певні аспекти виробництва комп'ютерних ігор, такі як створення та підбір музики і звукових ефектів, озвучення та захоплення руху, часто вимагають великих і не завжди здійснених фінансових вкладень, що виходять за межі можливостей розробника (можливо, через брак належного фінансування) (це актуально лише в тому випадку, якщо в компанії є відділ, відповідальний за ці операції). Оскільки компанії не вигідно наймати працівників для виконання цих завдань, більшість розробників вдаються до аутсорсингу, коли частина послуг проекту передається співвиконавцям.

Плани аутсорсингу оцінюються на етапі підготовки виробництва. Потім розраховуються часові та фінансові витрати на роботу поза межами компанії-розробника.

Однак деякі модульні інструменти, такі як відеокомпресори та редактори рівнів, можуть бути передані на аутсорсинг іншим студіям для розробки.

Вартість виробництва музичного твору залежить від його тривалості, методу виробництва (синтетичний чи живий) та досвіду композитора. Наприклад, у 2003 році вартість виробництва однієї хвилини високоякісної синтезаторної

музики становила сотнями та тисячами доларів США, а 60 хвилин музики для 20-годинної гри обійшлися б мовнику в сотні тисяч доларів США.

Озвучування як частина виробництва вимагає спеціальних і вузькоспеціалізованих навичок, що робить його придатним для аутсорсингу. Лише найбільші телерадіокомпанії мають у своєму штаті дикторів.

Оскільки студії захоплення руху є дорогими і їх важко побудувати, менші компанії навряд чи матимуть власні студії, і їм краще скористатися послугами аутсорсингових компаній.

1.8 Ігровий движок (фреймворк)

Ігрові фреймворки - це програмні комплекси, призначені для спрощення розробки ігор за допомогою вбудованих інструментів, таких як виведення графіки, пошук шляхів, перетворення 3D-редактора в ігрову модель, фізичні розрахунки, розрахунки зіткнень, оптимізація ресурсів процесора тощо.

Ігрові фреймворки відрізняються архітектурно, але зазвичай розробляються на основі компонентної архітектури, яка дозволяє замінювати або розширювати певні системи рушія більш спеціалізованими компонентами. Існує велика кількість ігрових рушіїв, кожен з яких має свої особливості та переваги. Деякі з них представлені тут:

- Unity: популярне середовище для розробки ігор у 3D та 2D;
- Unreal Engine: ігровий рушій, який використовується для створення багатьох великих ігор;
- Construct 3: ігровий рушій, який дозволяє створювати ігри без знань програмування;
- GameMaker: Ігровий рушій, який дозволяє створювати 2D-ігри.

У рамках даної роботи вибір зроблений на користь Unity, так як даний фреймворк є найбільш доступним і призначений для розробки на багатьох ігрових

платформах: Windows, Mac, Linux, Android, IOS, PlayStation 4, PlayStation 5, WebGL (для браузерів) тощо.

1.9 Особливості ігрового движка Unity

Unity має багато інструментів та функцій, які дозволяють розробникам створювати ігри та додатки різного типу та складності. Наприклад, Unity має вбудований редактор сцен, який дозволяє користувачам створювати та налаштовувати ігрові об'єкти, анімацію, світло і тіні, а також додавати звукові ефекти та музику.

Ігровий рушій підтримує такі мови програмування, як C# (рідна мова), C++, JavaScript та Boo, що дозволяє розробникам обирати найбільш підходящу мову для реалізації своїх ідей.

Крім того, Unity має потужний фізичний рушій, який дозволяє створювати реалістичні фізичні ефекти, такі як гравітація, зіткнення та рух об'єктів.

Unity також надає набір інструментів для створення багатокористувацьких ігор та додатків, включаючи мережеві можливості та інтеграцію з платформами соціальних мереж.

В Unity всі ігрові об'єкти будуються на трьох основних блоках: GameObjects, компоненти та змінні.

Кожен об'єкт у грі - це GameObject, і для того, щоб об'єкт працював, йому потрібні різні властивості, додані за допомогою компонентів.

Unity також має потужний фізичний рушій, який дозволяє створювати реалістичні фізичні ефекти, такі як гравітація, зіткнення та рух об'єктів.

Unity має вбудований редактор сцен, який дозволяє створювати та налаштовувати ігрові об'єкти, анімацію, світло та тіні, а також додавати звукові ефекти та музику. Даний редактор показано на рисунку 1.1.

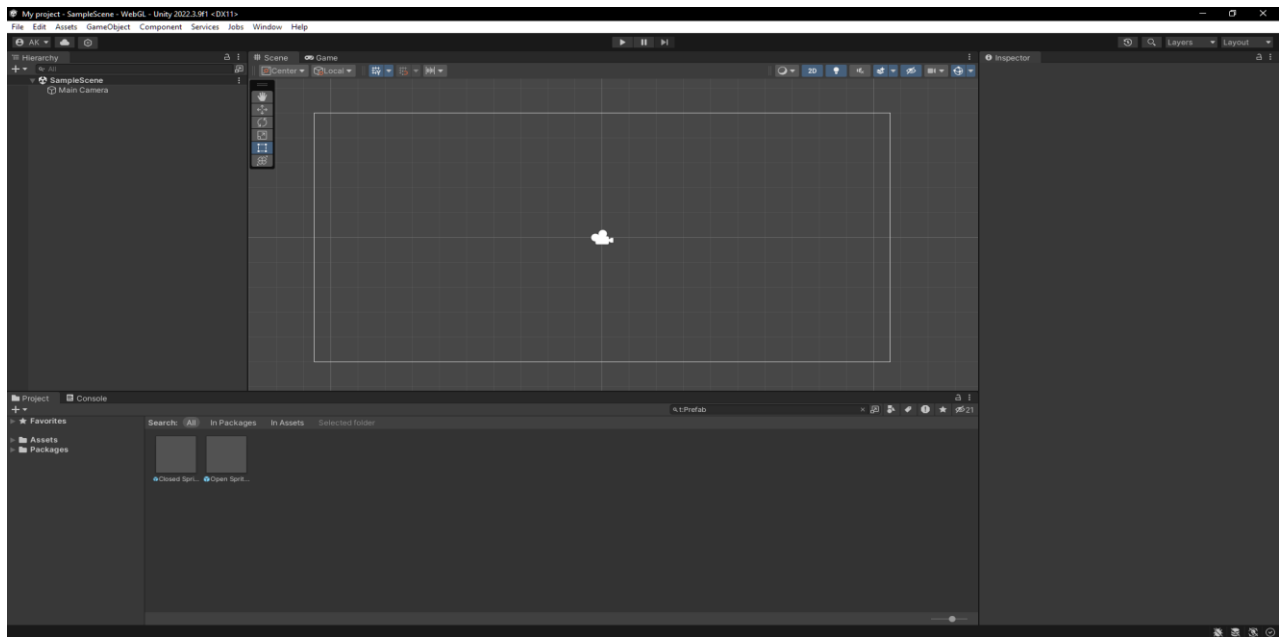


Рисунок 1.1 – Редактор ігрових сцен

Крім того, Unity надає набір інструментів для створення багатокористувацьких ігор та додатків, включаючи мережеві функції та інтеграцію з платформами соціальних мереж.

У рамках дипломної роботи бакалавра аркадна спортивна 2D гра «Ice Blitz» розроблюється для браузерів.

Браузерні ігри розробляються з використанням технології WebGL, яка дозволяє іграм і додаткам, створеним в Unity, запускатися у веб-браузері без необхідності встановлення додаткових плагінів. Щоб створити WebGL-збірку в Unity, обирається відповідна платформу в налаштуваннях проекту. Після створення WebGL-додатку в Unity можлива його публікація на веб-сайті або запуск на локальному сервері.

Unity підтримує API WebGL 1.0 та WebGL 2.0.

WebGL підтримується багатьма сучасними веб-браузерами, включаючи Google Chrome, Mozilla Firefox, Apple Safari, Microsoft Edge, Opera, Yandex Browser та UC Browser.

Однак, використання даної технології в Unity може спричинити проблеми з продуктивністю та сумісністю у різних браузерах та пристроях. Наприклад,

деякібраузери можуть не підтримувати WebGL, а мобільні пристрої можуть мати проблеми з продуктивністю.

При використанні WebGL також важливо враховувати специфіку використання JavaScript і API WebGL, які застосовуються для створення інтерактивної 3D комп'ютерної графіки, а також специфіку використання звуку, оскільки не всі формати аудіофайлів підтримуються WebGL.

WebGL використовує JavaScript API для відображення комп'ютерної інтерактивної 3D і 2D графіки у веб-браузері без необхідності встановлення додаткових плагінів. Додатки WebGL складаються з керуючого коду, написаного на JavaScript, і коду шейдерів (GLSL), що виконується на графічному процесорі (GPU) комп'ютера.

Оскільки API WebGL використовує математичні перетворення для створення 3D-об'єктів, важливо знати основні математичні поняття, що використовуються у 3D-графіці, щоб використовувати API WebGL.

Unity використовує FMOD для керування звуком на платформі, але оскільки FMOD використовує потоки, які не підтримуються у WebGL, у WebGL підтримується лише базова функціональність. Натомість Unity використовує реалізацію на основі вбудованого Web Audio API, що дозволяє браузеру виконувати відтворення та мікшування аудіо.

Слід також зазначити, що нова політика автовідтворення Google Chrome може перешкоджати автоматичному відтворенню аудіо та відео, наприклад, коли фонові музика автоматично відтворюється при завантаженні гри. Для отримання додаткової інформації про те, як увімкнути або вимкнути цю політику, див. документацію про політику автовідтворення Google Chrome.

2 РОЗРОБКА ГРИ

2.1 Написання дизайн-документа

Зазвичай перед тим, як почати розробку, особливо якщо вона з великим масштабом та бюджетом, складається дизайн-документ (диздок), у якій міститься уся інформація того, що буде з себе представляти гра.

Підготовкою цього документа займається гейм-дизайнер, який вирішує, як буде виглядати кінцева програма, призначена для розваги.

За книгою американського гейм-дизайнера, який приймав участь у таких проектах як Pac-Man World, God of War та Darksiders [8] пропонується шаблон, який складається з 10 сторінок.

2.1.1 Шаблон десятисторінника від Скота Роджерса

2.1.1.1 Сторінка 1

Загальна інформація:

- назва;
- платформи;
- цільова аудиторія;
- віковий рейтинг;
- дата публікації/видання, що планується;
- конкуренти та натхненники;
- титульна картинка, концепт-арт;
- контактні дані.

2.1.1.2 Сторінка 2

Короткий опис сюжету у декілька параграфів. Може містити хто є героєм, який вид камери (від першого або третього обличчя), жанр гри (шутер, платформер, рольова гра тощо), де відбуваються події, основна мета, опис списку локацій та особливості геймплея (ігрового процесу).

2.1.1.3 Сторінка 3

Опис персонажа, його минуле, реакції на складнощі, типаж тощо. Бажано вказувати лише подробиці, що впливають на сюжет, атмосферу та геймплей.

2.1.1.4 Сторінка 4

Геймплей. Опис ігрового процесу, його можливості. Чи поділяється гра на рівні? Які їх особливості?

Можливі використання діаграм та концепт-артів для зображення складних елементів, опис технічних обмежень та особливостей використання платформи, а також наявність можливості грати з іншими гравцями.

2.1.1.5 Сторінка 5

Ігровий світ. Описи та зображення світу. Список локацій. Короткі описи того, що гравець зможе знайти. Якій настрій передається у кожных місця? Яка музика використовується? Як локації з'єднані між собою?

2.1.1.6 Сторінка 6

Ігровий досвід. Що бачить гравець при старті? Який настрій повинна створювати гра? Як музика та звуки впливає на атмосферу? Яким чином здійснюється навігація?

2.1.1.7 Сторінка 7

Ігрова механіка – правило або набір правил взаємодії гравця з грою.

Приклади механік:

Можливі небезпеки. Для документації даного рівня достатньо трьох для прикладу. Наприклад, лава, шипи та ями.

Предмети, бонуси, які підсилюють або допомагають пройти гру. Наприклад, аптечки, броня, набой.

Колекційні предмети. Ними можуть бути всілякі трофеї, картки, шматки пазлу, записки тощо. Їх вплив збір на гру. Можливі секрети, що відкриваються, такі, як ігрові режими або рівні.

Якщо є система економіки, то опис йде у загальних рисах. Є можливим короткий опис магазинів у грі.

2.1.1.8 Сторінка 8

Вороги, які гравець зустрічає, їх унікальність та методи боротьби з ними. В них, як правило, наявний штучний інтелект. Особливо стосується так званих босів, особливих ворогів, які з'являються у конкретних локаціях, мають унікальний набір атак та тактика бою супротив них.

2.1.1.9 Сторінка 9

Сюжетні ролики. Їх наявність. Як вони будуть показуватись? Якщо є відсилення, вказати їх.

2.1.1.10 Сторінка 10

Бонусні матеріали. Що замотивує гравця продовжувати грати? Нові режими або «Нова гра+». Або завантажуваний контент. Або якісь особливі події. Інші можливості.

2.1.2 Дизайн-документ гри «Ice Blitz»

Назва: «Ice Blitz»

Платформа: Браузери з підтримкою WebGL

Натхненники: Ice Hockey (NES), Blades of Steel (NES)

Жанр: 2D спортивна аркада

Титульний екран гри показано на рисунку 2.1.



Рисунок 2.1 – Титульний екран

2.1.2.1 Короткий опис сюжету

В одному з університетів Канади відбуваються щорічні змагання з хокею. Команди "Sapphire Storm", "Ruby Raiders", "Emerald Inferno", "Amethyst Avalanche" та "Topaz Tornado" наразяться жорсткому протистоянню за кубок найсильніших. Хто з них зможуть перемогти у цьому заході?

Кубок показано на рисунку 2.2.



Рисунок 2.2 – Кубок найсильніших

2.1.2.2 Команди

Гравці вище описаних команд будуть мати різноманітний склад, який буде прихилити до певного стилю гри. Наприклад, з нахилом на швидкість або силу, з нахилом на захист чи напад. Перелік та вигляд команд показано на рисунку 2.3.

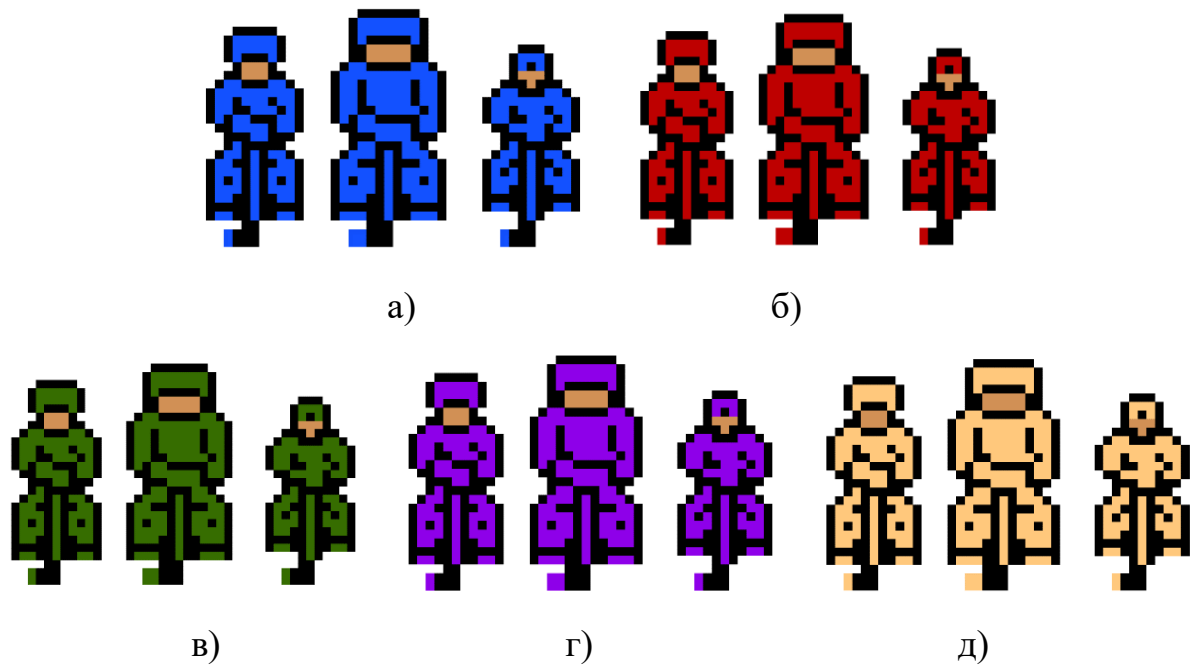


Рисунок 2.3 – Гравці команд: а) "Sapphire Storm"; б) "Ruby Raiders"; в) "Emerald Inferno"; г) "Amethyst Avalanche"; д) "Topaz Tornado"

Візуальна та аудіо складова виконана у стилі 8-бітних ігор як на приставках «Денді» (Nintendo Entertainment System/Family Computer)

2.1.2.3 Ігровий процес

За 10 хвилин, які відведені на матч, одна з заздалегідь обраних протиборчих команд, забиваючи голи, повинна отримати якомога більше очок для перемоги.

Команди буде складатися з 3-5 гравців та воротаря та будуть мати такі параметри як швидкість, точність ударів та фізична сила для збиття з ніг суперника в залежності від статури та позиції на полі. Наприклад, одні гравці можуть сильніше у фізичній силі, але програвати у швидкості, інші можуть точно наносити удари, але збивати з ніг будуть не дуже сильно, ті, хто в нападі, наносять удари більш точно, а навпаки - навмання.

На полі будуть періодично з'являтися бонуси, які зможуть впливати на хід матчу на користь одної з сторін, наприклад, підвищена швидкість або імунітет до нокдауну. Як приблизно виглядатиме поле показано на рисунку 2.4.

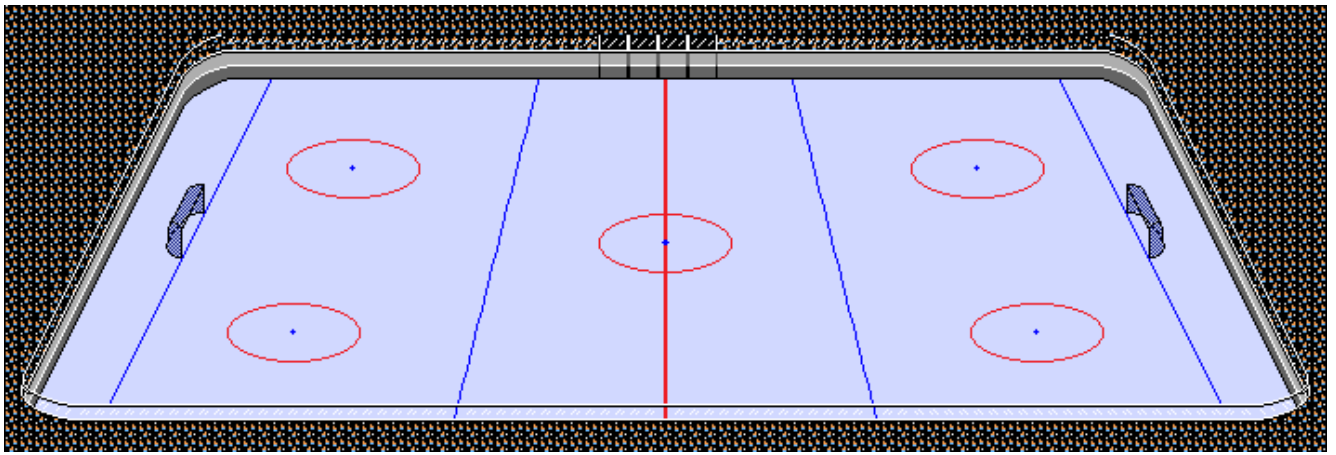


Рисунок 2.4 – Хокейне поле

2.1.2.4 Інтерфейс

Інтерфейс буде переважати мінімалізмом. На стартовому екрані буде вибір між одним гравцем проти штучного інтелекту або між двома гравцями друг проти друга та можливість вивчити управління, котрий буде складатися з напрямів переміщення, пасами, ударами та збиття опонента з ніг. Далі необхідно буде обрати команду та розташування гравців після чого гра почнеться. Табло з таймером та очками, набраними командами за голи, буде розташований у верхній частині екрана.

2.1.2.5 Ігрові механіки

Витривалість: важлива складова для гри для здійснення певних фізичних дій. У разі якщо вона досягне нульового показника, буде накладатися штраф на швидкість та час нокдауна.

Переміщення: гравці окрім ковзання по льоду зможуть використовувати ривки (dash) та спринт (sprint) для мобільності. Ці можливості будуть витрачати витривалість (stamina), спринт – поступово, а ривки – фіксовану кількість.

Бойова система: учасники зможуть відбирати або перехоплювати шайби та шляхом штовхання збивати з ніг суперника. Остання можливість буде витрачати певну кількість витривалості.

Бонуси:

- невразливість до нокдауну на n секунд. Виконаний у вигляді шолома;
- прискорення у x рази на y секунд. Виконаний у вигляді ковзана;

- потужний удар збиває гравців на p секунд з ніг окрім воротаря.
Виконаний у вигляді ключки;

- підсилене штовхання: збитий з ніг буде у нокдауні у x рази. Виконаний у вигляді боксерської рукавички.

Розташування на полі будуть залежати від типу бонуса та з'являтися періодично у момент підбору

Вигляд вище обчислених бонусів показані на рисунку 2.5.

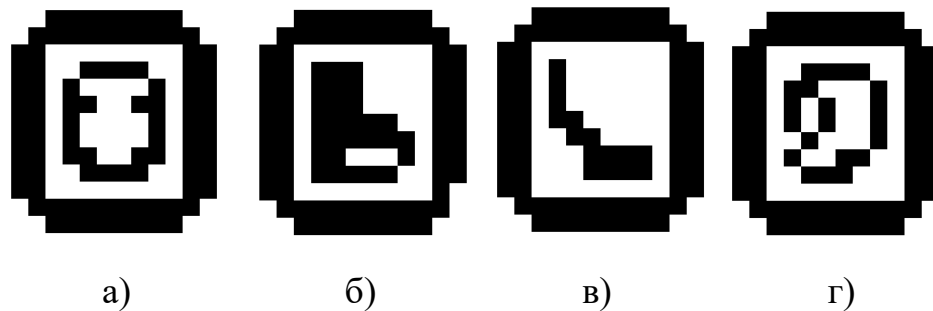


Рисунок 2.5 – Бонуси: а) невразливість; б) прискорення;
в) потужний удар; г) підсилене штовхання

2.2 Створення візуальної частини

Створення візуальної частини гри є одним з найважливіших аспектів, оскільки саме графіка багато в чому визначає перше враження гравця. Цей процес включає кілька ключових етапів:

- концептуальний дизайн: на цьому етапі створюються концептуальні ескізи та дизайн-проекти, що визначають загальний стиль гри. Художники та дизайнери працюють над створенням образів персонажів, середовища та об'єктів. Концептуальний дизайн включає розробку кольорових схем, форм та текстур, які будуть використовуватись у грі;

- 3D-модельовання та анімація: створені концептуальні ескізи перетворюються на тривимірні моделі. Використовуються програми для 3D-

моделювання, такі як Blender або Maya, для створення персонажів, об'єктів та середовища. Аніматори створюють рухи та анімації для персонажів та об'єктів, забезпечуючи реалістичність та динамічність гри. Анімація включає рухи персонажів, взаємодії з об'єктами та спецефекти;

- текстурування та освітлення: на цьому етапі створюються текстури для моделей, що додають деталізації та реалізму. Освітлення використовується для створення атмосфери та підкреслення важливих елементів гри. Важливо також налаштувати тіні, відображення та інші ефекти, що роблять картинку більш живою та насиченою;

- оптимізація: важливо оптимізувати візуальні елементи для забезпечення плавної роботи гри на різних пристроях. Це включає зменшення кількості полігонів, оптимізацію текстур та ефектів. Використання LOD (Level of Detail) дозволяє зменшувати деталізацію об'єктів на великій відстані, що значно знижує навантаження на графічний процесор. Створення візуальної частини гри потребує тісної співпраці між художниками, дизайнерами та програмістами. Важливо забезпечити високу якість графіки, що не тільки візуально приваблює, але й не створює надмірного навантаження на систему, забезпечуючи плавний геймплей;

- інтеграція та тестування: після завершення створення моделей, текстур та освітлення, їх необхідно інтегрувати в гру. Цей етап включає в себе розміщення моделей у відповідній геометрії гри, налаштування освітлення та ефектів, а також додавання анімацій до персонажів та об'єктів у грі. Після інтеграції візуальних елементів проводиться тестування для виявлення та виправлення будь-яких помилок, артефактів або несправностей в графіці. Це може включати в себе тестування різних конфігурацій графічного обладнання та різних роздільних здатностей екрану для забезпечення сумісності та оптимальної продуктивності на різних пристроях;

- полірування та оптимізація: після тестування графічні елементи можуть потребувати додаткового полірування, щоб забезпечити високу якість та зрозумілість. Це може включати виправлення дрібних дефектів, оптимізацію швидкості завантаження текстур або видалення зайвих деталей. Оптимізація

графічних ефектів і ресурсів може бути важливою для забезпечення оптимальної продуктивності гри на різних пристроях, особливо на мобільних платформах або старіших комп'ютерах. Це може включати в себе використання технік компресії текстур, які дозволяють зменшити їх розмір без значного втрати якості, а також оптимізацію рендерингу, щоб уникнути зайвого навантаження на графічний процесор;

- підтримка після випуску: після випуску гри важливо забезпечити підтримку візуальної частини на протязі її життєвого циклу. Це може включати в себе випуск патчів для виправлення графічних проблем, а також вдосконалення візуальних ефектів або додавання нового контенту для поліпшення враження від гри. Підтримка також може включати розробку версій гри для нових платформ або оновлення графічного движка для забезпечення сумісності з новим апаратним забезпеченням;

- дизайн інтерфейсу користувача (UI): окрім самої графіки, важливо також розробляти інтерфейс користувача, який буде чітким, зрозумілим та привабливим. Це включає в себе створення елементів інтерфейсу, таких як меню, кнопки, іконки, шрифти та інші важливі компоненти, які сприяють зручності гри. Дизайн UI повинен бути гармонійно вписаним у загальний стиль гри та легко сприйманим для гравців будь-якого рівня досвіду;

- візуальні ефекти: додавання спецефектів до гри може значно підвищити її візуальну привабливість та динаміку. Це можуть бути ефекти погоди (дощ, сніг, туман), вибухи, освітлення, частинки (дим, вогонь, іскри) та багато інших. Реалістичні візуальні ефекти додають атмосферності грі та підсилюють враження від ігрових подій;

- адаптація до різних пристроїв та платформ: з урахуванням різних характеристик графічного обладнання та екранів різних пристроїв, важливо розробляти візуальну частину гри з урахуванням їх можливостей та обмежень. Наприклад, на мобільних пристроях може бути обмежена потужність графічного процесора, тому графічні ефекти можуть бути менш деталізованими або відключеними, щоб забезпечити плавну роботу гри;

- культурний та етнічний контекст: у випадку, коли гра призначена для міжнародного ринку, важливо також враховувати культурні та етнічні відмінності у візуальних представленнях. Кольори, символи, стилі та образи можуть мати різні значення та сприйняття в різних країнах та культурах. Тому дослідження та врахування цих аспектів може допомогти створити візуальну частину гри, яка буде привабливою та зрозумілою для гравців з усього світу;

- стійкість до змін: під час розробки гри можуть змінюватися вимоги до візуальної частини, а також технічні обмеження. Тому важливо створювати графічний контент таким чином, щоб він був легко модифікованим та адаптованим до змін, що можуть виникнути протягом розробки гри.

2.3 Написання музики та звукових ефектів

Музика та звукові ефекти відіграють важливу роль у створенні атмосфери гри та підвищенні емоційного впливу на гравця. В процесі розробки аудіо супроводу необхідно враховувати жанр гри, її сюжет та геймплей. Музика повинна доповнювати візуальну складову та підсилювати емоційне сприйняття гри.

Процес створення аудіо включає кілька етапів:

- концептуальне проектування: аналіз сюжету та настрою гри для визначення потрібного музичного стилю, створення основних музичних мотивів, які відображають ключові емоції гри. та визначення основних звукових ефектів, необхідних для підсилення занурення;

- запис та синтез: вибір інструментів для запису музики, з урахуванням їхнього відповідності стилеві гри, створення або вибір звуків синтезаторів, що відповідають вимогам гри та синхронізація музики з ігровими подіями для створення плавного переходу між різними частинами гри;

- мікшування та обробка: застосування ефектів реверберації, щоб створити відчуття простору та глибини, еквалізація для забезпечення балансу та чистоти звуку на різних аудіодоріжках та додавання інших звукових ефектів, таких як ехо, фільтри, для підвищення емоційного впливу;

- інтеграція в гру: налаштування тригерів для звукових ефектів, щоб вони відтворювалися у відповідь на конкретні дії гравця або події в грі, встановлення системи динамічного аудіо, яка змінює музичний фон залежно від ситуації у грі та тестування для переконання, що аудіо ефективно підсилює занурення та емоційний досвід гравця;

- тестування та звукова оптимізація: проведення аудіо-тестування для перевірки якості звуку на різних аудіосистемах та пристроях, коригування гучності та балансу звуку для забезпечення оптимального досвіду гравця та оптимізація аудіо для максимальної ефективності роботи з ігровим движком та обмеженими ресурсами пристроїв;

- підтримка та оновлення: надання підтримки щодо аудіо-проблем та оновлення звукового супроводу після випуску гри, вирішення звукових багів та покращення якості звуку на основі зворотного зв'язку від гравців та додавання нових аудіо-контенту та оновлень, які відповідають новим вимогам та трендам у галузі геймдеву.

Ці етапи допомагають не лише створити аудіо-супровід, який відповідає вимогам гри, але й підтримувати та вдосконалювати його протягом усього життєвого циклу продукту.

2.4 Складання ігрового процесу

2.4.1 Діаграми варіантів використання

Діаграма використання дозволяє чітко відобразити, як користувачі взаємодіють з системою та які функції вони використовують. Вона ілюструє

конкретні потреби та дії користувачів, що допомагає у процесі проектування та розробки.

Визначення потреб користувачів: діаграма дозволяє визначити, які саме потреби мають користувачі системи, що дає можливість розробникам краще розуміти, які функції необхідно реалізувати.

Встановлення пріоритетів: за допомогою діаграми можна встановити, які функції є найбільш важливими для користувачів, що дозволяє розробникам зосередити зусилля на їх реалізації першочергово.

Покращення комунікації: діаграма використання є ефективним інструментом для комунікації з різними зацікавленими сторонами, такими як розробники, менеджери та клієнти, оскільки вона проста у використанні та зрозуміла для всіх.

Документування процесу: використання діаграми допомагає документувати процеси, що відбуваються в системі, що є корисним для подальшого аналізу та удосконалення. Приклад однієї з них показано на рисунку 2.6.

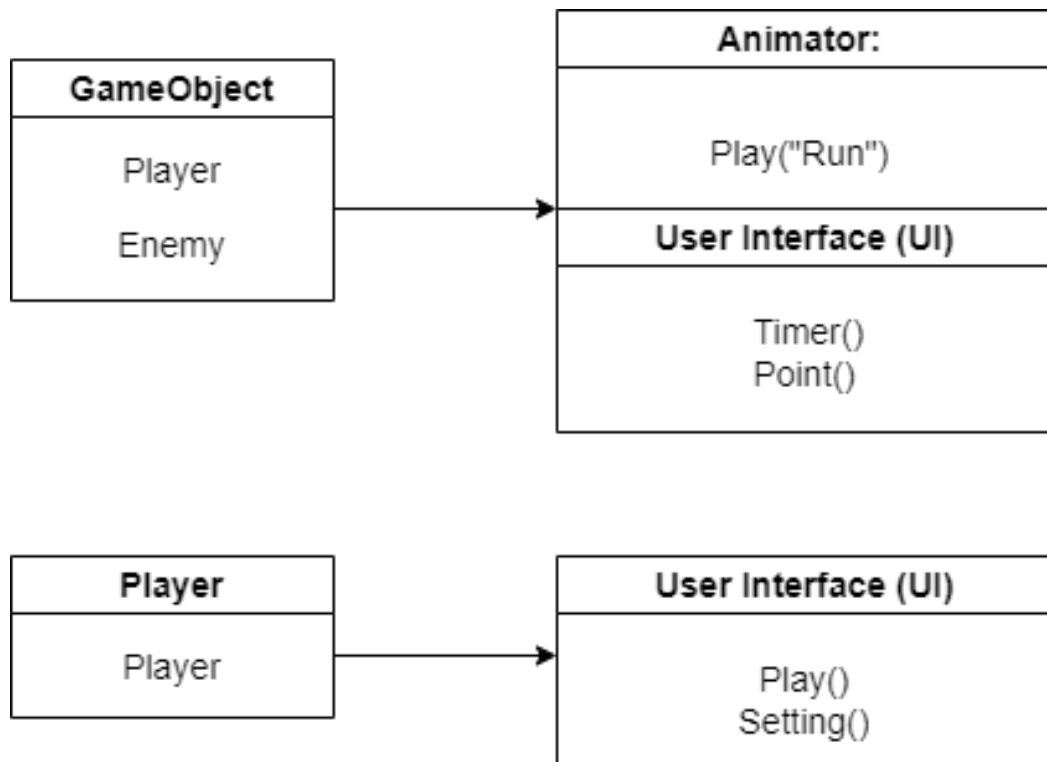


Рисунок 2.6 – Діаграма варіантів використання

2.4.2 Діаграми послідовності

Діаграма послідовності - це графічне зображення послідовності взаємодії між об'єктами у системі. Вона відображає порядок виконання різних операцій або викликів методів в межах конкретного сценарію взаємодії.

Візуалізація взаємодії об'єктів: діаграма допомагає візуалізувати, як саме об'єкти взаємодіють між собою, що полегшує розуміння складних процесів у системі.

Аналіз логіки виконання: вона дозволяє аналізувати логіку виконання операцій та виявляти можливі проблеми або оптимізувати процеси.

Документування бізнес-процесів: діаграми послідовності часто використовуються для документування бізнес-процесів, що забезпечує краще розуміння процесів для всіх учасників проекту.

Підвищення якості проекту: діаграми послідовності допомагають виявити потенційні помилки та недоліки ще на етапі проектування системи. Вони дозволяють заздалегідь спрогнозувати можливі проблеми та вирішити їх до того, як вони стануть критичними. Це сприяє підвищенню якості розроблених продуктів або систем.

Спрощення комунікації: діаграми послідовності є ефективним засобом комунікації між членами команди проекту, клієнтами та іншими зацікавленими сторонами. Вони дозволяють уникнути непорозумінь щодо логіки взаємодії об'єктів та операцій, спрощуючи спілкування та забезпечуючи збільшення продуктивності у всьому проекті.

Вдосконалення процесів: аналізуючи діаграму послідовності, команди можуть ідентифікувати можливі шляхи оптимізації процесів взаємодії між об'єктами. Це дозволяє покращити продуктивність та ефективність системи, зменшити витрати часу та ресурсів на виконання операцій. Загалом, діаграма послідовності виявляється незамінним інструментом як для розробників програмного забезпечення, так і для бізнес-аналітиків, оскільки вона сприяє розумінню, аналізу та вдосконаленню процесів у будь-якій сфері діяльності.

Приклад діаграми послідовності показано на рисунку 2.7.

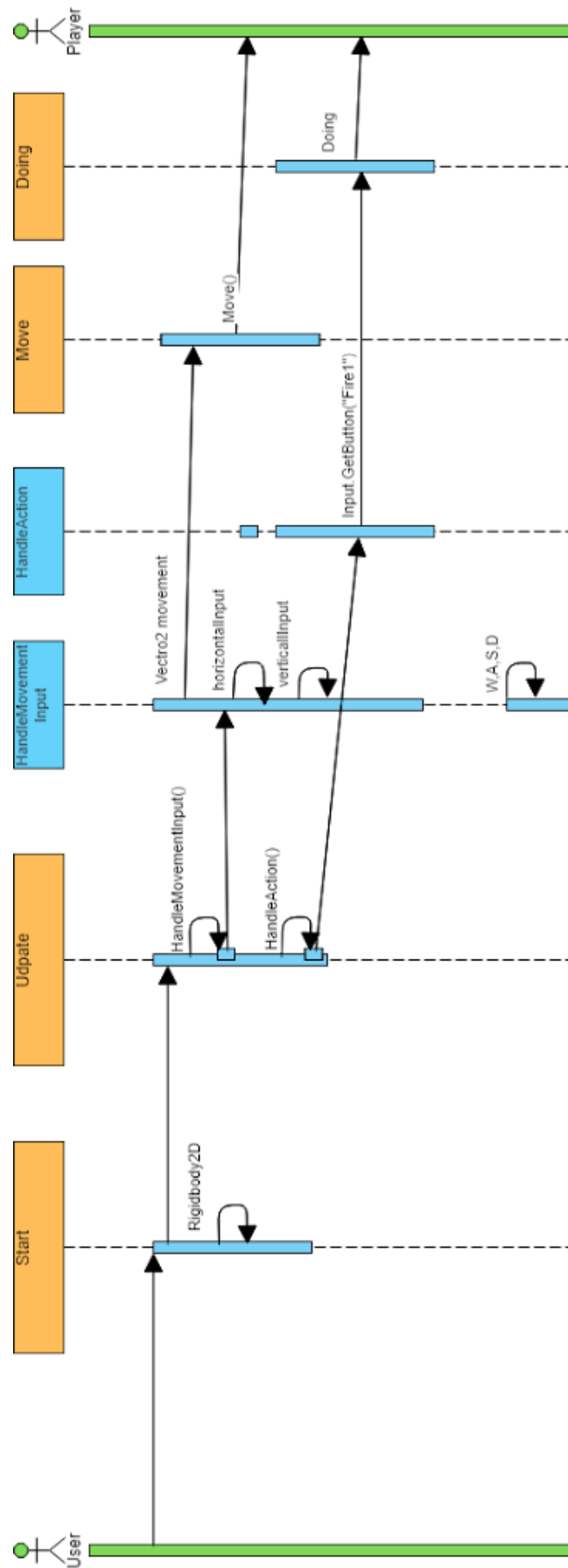


Рисунок 2.7 – Діаграма послідовності

2.4.3 Особливості WebGL в UNITY

Unity - це популярний інтегрований середовище розробки для створення ігор та інтерактивних додатків. WebGL (Web Graphics Library) - це технологія, що дозволяє запускати графічно потужні додатки у веб-браузері без необхідності встановлення додаткових плагінів. Ось деякі особливості WebGL в Unity:

- веб-орієнтовані можливості: WebGL дозволяє створювати ігри та додатки, які можуть бути запуснені у будь-якому сучасному веб-браузері без необхідності встановлення додаткового програмного забезпечення. Це значно розширює аудиторію користувачів, оскільки будь-хто з доступом до браузера може скористатися додатком. Підтримка різних браузерів, таких як Chrome, Firefox, Safari та Edge, забезпечує крос-браузерну сумісність, що дозволяє досягти ширшого охоплення користувачів;

- оптимізована продуктивність: Unity оптимізує генерацію коду та роботу фреймворка для забезпечення ефективної роботи ігор та додатків на платформі WebGL. Це включає оптимізацію роботи з пам'яттю, зменшення розміру завантажуваних ресурсів та покращення швидкості виконання. Використання технологій стиснення даних, таких як Gzip або Brotli, дозволяє зменшити час завантаження додатків, що покращує користувацький досвід;

- підтримка графіки: WebGL в Unity підтримує передові графічні можливості, такі як шейдери, освітлення, текстури і спеціальні ефекти. Це дозволяє створювати вражаючі візуальні ефекти та реалістичні сцени, що підвищує привабливість ігор та додатків. Використання шейдерної мови GLSL (OpenGL Shading Language) дозволяє розробникам створювати складні графічні ефекти, такі як реалістичні тіні, відблиски та рефлекси;

- крос-платформеність: додатки, розроблені з використанням Unity та платформи WebGL, можуть запускатися на різних пристроях та операційних системах, що робить їх крос-платформовими та доступними для широкого кола користувачів. Це включає десктопи, ноутбуки, планшети та смартфони. Така універсальність дозволяє розробникам створювати додатки, які одночасно підтримують декілька платформ, що зменшує витрати на розробку та підтримку;

- підтримка веб-технологій: WebGL у Unity підтримує використання веб-технологій, таких як HTML, CSS та JavaScript, для створення інтерфейсу користувача та взаємодії зі сторонніми веб-сервісами. Це дозволяє інтегрувати додатки з іншими веб-ресурсами та API. Використання WebAssembly (Wasm) для підвищення продуктивності виконання коду, що забезпечує більш швидке та ефективне виконання веб-додатків;

- вимоги до продуктивності: важливо враховувати, що платформа WebGL може мати певні обмеження щодо продуктивності, зокрема щодо обробки графіки та виконання складних обчислень у порівнянні зі стандартними десктопними ігровими платформами. Це означає, що для досягнення оптимальної продуктивності можуть бути необхідні певні компроміси у складності графіки або функціональності. Оптимізація ресурсів, таких як текстур та моделі, а також зменшення кількості обчислень, які виконуються на стороні клієнта, допомагає покращити продуктивність;

- безпека: WebGL забезпечує високий рівень безпеки, оскільки працює у межах браузера та використовує механізми захисту браузера від потенційно небезпечних дій. Це включає ізоляцію контенту та захист від міжсайтових скриптів (XSS). Підтримка протоколів безпеки, таких як HTTPS, для захищеної передачі даних між сервером та клієнтом;

- розширення можливостей розробки: Unity дозволяє використовувати різні плагіни та бібліотеки для розширення функціональності WebGL додатків. Це може включати інтеграцію з аналітичними інструментами, системами монетизації або соціальними мережами. Інструменти для відлагодження та моніторингу продуктивності, такі як Chrome DevTools або Firefox Developer Tools, допомагають розробникам виявляти та виправляти помилки, а також оптимізувати додатки для кращої продуктивності.

Врахування цих особливостей платформи WebGL в Unity дозволяє ефективно розробляти ігри та інтерактивні додатки, які можуть бути запуснені безпосередньо у веб-браузері, забезпечуючи широку доступність та можливість

використання на різних пристроях і платформах. Необхідні налаштування показані на рисунках 2.8-2.9

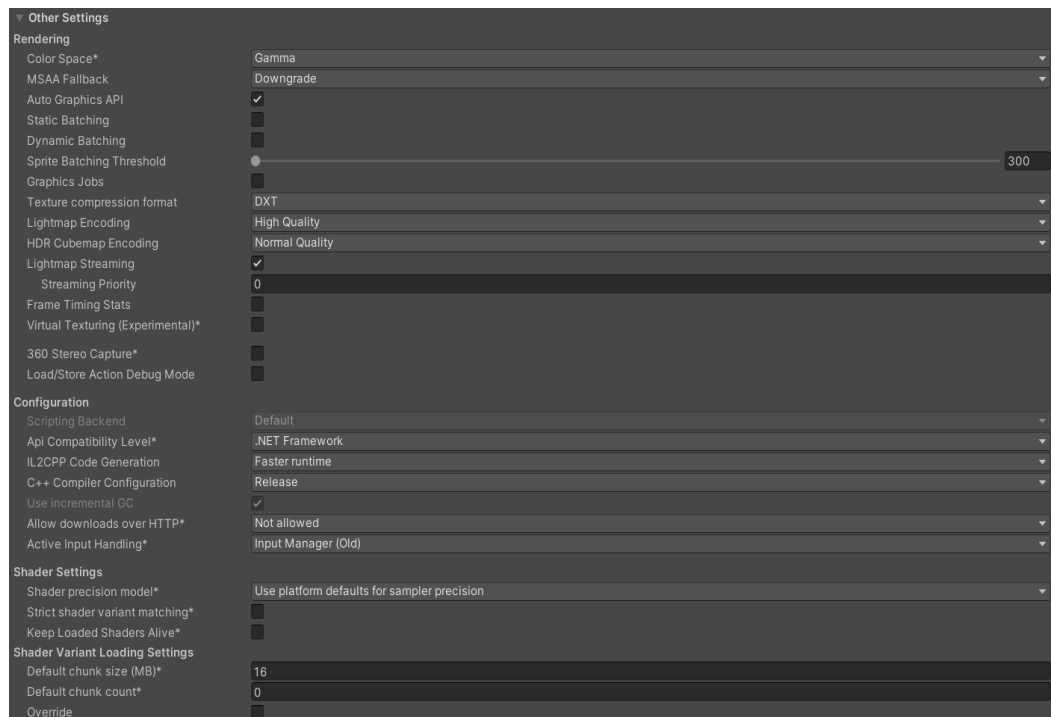


Рисунок 2.8 – Налаштування проекту для отримання успішного проекту який підтримує WebGL

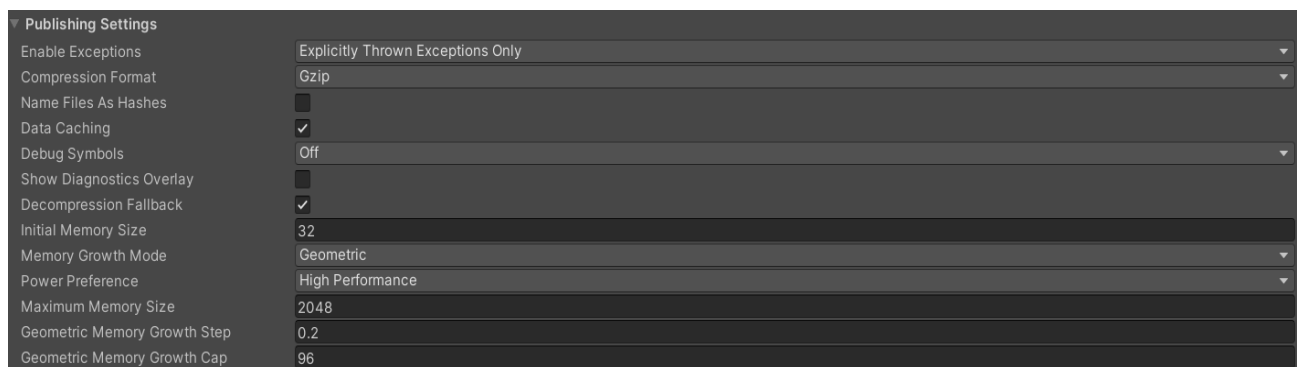


Рисунок 2.9 – Налаштування публікації проекту для відображення на локальному сервері або сайті

2.4.4 Створення інтерфейсу

Створення інтерфейсу є важливою частиною проектування інтерфейсу користувача для браузерної гри на Unity для платформи WebGL. Для цього потрібно виконати наступні кроки:

- визначення необхідних елементів: визначити, які елементи необхідно включити до меню. Це можуть бути такі елементи, як кнопки для початку нової гри, завантаження збереженої гри, перегляду налаштувань, виходу тощо;

- розташування елементів: розташувати елементи меню так, щоб вони були легко доступні та зрозумілі для користувача. Зазвичай кнопки розміщуються по центру екрану для кращої видимості та зручності;

- додавання анімаційних ефектів: додавати анімаційні ефекти та плавні переходи між елементами меню, щоб зробити його більш динамічним та привабливим для користувача;

- налаштування параметрів гри: забезпечити можливість користувачам налаштовувати різні параметри гри, такі як рівень гучності звуку, через меню налаштувань;

- тестування інтерфейсу: провести тестування інтерфейсу головного меню з різними типами користувачів, щоб переконатися, що він є зручним та легким у використанні. Зворотній зв'язок від користувачів допоможе виявити недоліки та внести необхідні корективи;

- доступність інтерфейсу: врахування можливостей людей з обмеженими можливостями, такими як користувачі з вадами зору або дефіцитом слуху. Забезпечення можливості налаштування розміру шрифтів, контрастності кольорів та інших параметрів для полегшення доступу до гри для всіх користувачів;

- локалізація інтерфейсу: переклад текстів інтерфейсу на різні мови для забезпечення максимальної доступності для гравців з різних країн та культур. Включення можливості зміни мови інтерфейсу без перезапуску гри;

- підтримка різних роздільностей екрану: адаптація головного меню до різних роздільностей екрану, забезпечення його коректного відображення як на великих моніторах, так і на пристроях з невеликими екранами, такими як планшети або мобільні телефони;

- інтеграція з системними налаштуваннями: можливість автоматичної адаптації параметрів гри до налаштувань пристрою користувача, таких як мова системи, регіональні налаштування, наявність звукових пристроїв тощо;

- оптимізація для різних браузерів: перевірка роботи головного меню на різних браузерах та забезпечення його коректної роботи без проблем з відображенням чи функціональністю;

- інтерактивні елементи: додавання можливості взаємодії з елементами меню, наприклад, зміна кольору кнопок при наведенні курсору миші або відтворення звукового ефекту при натисканні на кнопку;

- підтримка різних типів вводу: забезпечення можливості навігації по меню як за допомогою миші, так і за допомогою клавіатури або екранної клавіатури для користувачів з обмеженими можливостями чи відсутнім доступом до миші.

Приклади інтерфейсів та їх архітектури показані на рисунках 2.10-2.14.



Рисунок 2.10 – Основний вигляд UI головного меню



Рисунок 2.11 – Основний вигляд UI ігрової сцени



Рисунок 2.12 – Основний вигляд UI меню управління грою

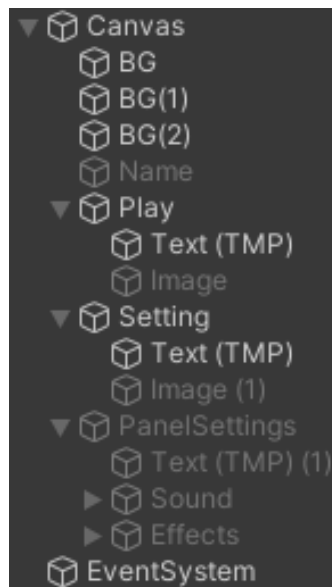


Рисунок 2.13 – Архітектура компонентів UI головного меню

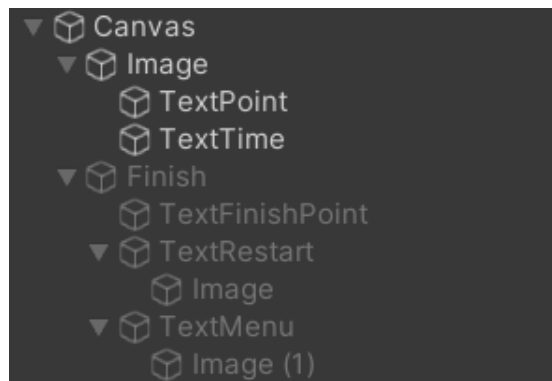


Рисунок 2.14 – Архітектура компонентів ігрової сцени

2.4.5 Як проектувати інтерфейс користувача

Планування і створення інтерфейсу користувача (UI) - це критичний етап в розробці будь-якої програми чи ігрового додатку. Ось кілька додаткових порад, які можна врахувати під час проектування UI:

- визначення необхідних елементів: основні елементи можуть включати не лише кнопки для початку нової гри або завантаження збереженої, але й доступ до налаштувань, інструкцій гри, можливості виходу тощо. Ретельно обдумайте, які саме функції будуть найбільш корисні та зрозумілі для вашої аудиторії;

- розташування елементів: елементи меню розташовуються так, щоб вони були легко доступні та зрозумілі для користувача. Зазвичай, кнопки розміщуються в центрі екрану або по краях зручного для користувача доступу;

- додавання анімаційних ефектів: анімаційні ефекти можуть додати динаміки та привабливості до вашого меню. Необхідно розглянути можливість використання плавних переходів між елементами меню, зміну кольорів кнопок при наведенні курсора, або інші ефекти, які привернуть увагу користувача та покращать його враження;

- налаштування параметрів гри: потрібно забезпечити можливість користувачам налаштовувати різні параметри гри через меню налаштувань. Це може включати регулювання рівня гучності звуку, яскравості чи роздільної здатності, вибір мови чи інших параметрів, які можуть покращити комфорт користування додатком;

- тестування з різними типами користувачів: проводиться тестування інтерфейсу головного меню з різними типами користувачів, включаючи тих, хто має різний рівень досвіду користування комп'ютером або іграми. Зібрані відгуки та спостереження допоможуть виявити потенційні проблеми та покращити зручність та ефективність інтерфейсу;

- врахування адаптивності: при проектуванні інтерфейсу необхідно враховувати, що додаток може використовуватися на різних пристроях з різними розмірами екранів та забезпечити адаптивність елементів інтерфейсу так, щоб вони були зручними як на десктопах, так і на мобільних пристроях;

- увага до візуального оформлення: важливо, щоб інтерфейс виглядав привабливо та професійно. Слід звертати увагу на вибір кольорів, шрифтів, іконок та інших візуальних елементів та використовувати сучасні дизайн-тренди, але не варто забувати про основні принципи зручності та читабельності;

- збереження консистентності: однаковий стиль та організацію елементів інтерфейсу підтримується на всіх етапах взаємодії з користувачем. Консистентність допомагає зменшити плутанину та покращує зрозумілість інтерфейсу;

- розділення на прості кроки: якщо ваш додаток має складний функціонал, розглядається можливість розділити процес використання на кілька простих кроків. Це допоможе користувачам легше зрозуміти, як використовувати ваш додаток;

- звернення до стандартів та паттернів дизайну: використання відомих стандартів та паттернів дизайну сприяє зручності та простоті використання. Можливе скористатися готовими бібліотеками або фреймворками, щоб прискорити процес розробки та забезпечити високу якість інтерфейсу;

- перевірка на доступність: важливо, щоб інтерфейс був доступним для всіх користувачів, включаючи людей з обмеженими можливостями. Врахуйте принципи доступності при проектуванні та розробці інтерфейсу;

- збір та аналіз відгуків користувачів: після випуску додатку не забувати збирати відгуки від користувачів та аналізувати їх. Це допоможе виявити слабкі місця інтерфейсу та зробити необхідні покращення для поліпшення користувацького досвіду.

Слід пам'ятати, що головне меню є першим контактом користувача з вашим додатком, тому важливо зробити його якомога привабливим, зручним та інтуїтивно зрозумілим.

2.4.6 Написання скриптів

У Unity, скрипт — це програмний код, написаний мовою програмування C#, який використовується для управління поведінкою об'єктів в ігровій сцені. Скрипти є невід'ємною частиною розробки ігор та інтерактивних додатків у Unity, оскільки вони дозволяють створювати логічні зв'язки та керувати різними аспектами гри, такими як рух персонажів, взаємодія з об'єктами, обробка подій та багато іншого.

Основні концепції скриптів у Unity:

- MonoBehaviour: більшість скриптів у Unity успадковуються від класу MonoBehaviour. Цей клас пропонує базові функції, які дозволяють інтегрувати скрипт в ігрову систему Unity;

- методи життєвого циклу: `Start()` - викликається один раз під час ініціалізації скрипта. Зазвичай використовується для налаштування початкового стану. `Update()` - викликається кожен кадр. Використовується для оновлення логіки гри, наприклад для обробки введення користувача або руху об'єктів. `FixedUpdate()` – викликається через фіксовані інтервали часу. Використовується до роботи з фізикою. `LateUpdate()` - викликається кожен кадр після `Update()`. Корисно для виконання логіки, яка має відбуватися після всіх оновлень;

- взаємодія з об'єктами: скрипти можуть взаємодіяти з компонентами об'єктів, до яких вони прикріплені, наприклад з трансформами, рендерерами і колайдерами;

- обробка введення користувача: Unity пропонує клас `Input`, який дозволяє обробляти введення з клавіатури, миші, тачпадів та інших пристроїв;

- використання компонентного підходу: Unity використовує компонентну архітектуру, де кожен об'єкт може мати багато різних компонентів (наприклад, рендерер, колайдер, скрипт). Скрипти можуть додавати та видаляти компоненти, а також взаємодіяти з ними;

- інспектор та параметри: змінні у скриптах можуть бути зроблені публічними.

Нижче приведений короткий опис кожного зі скриптів:

У лістингу 2.1 виконується керування поведінкою бота, який пересувається до шайби та намагається вдарити її в напрямку воріт.

Ініціалізація: в методі `Start` створюється прив'язка до компоненту `Rigidbody2D` бота. Також викликається метод `Invoke`, який затримує виклик методу `StartMovement` на випадковий час від 2 до 4 секунд.

Старт руху: метод `StartMovement` змінює значення змінних `moveSpeed`, `hitDistance` та `hitForce` на випадкові значення в заданих межах і встановлює `puckActive` в `false`, щоб бот почав рухатися.

Оновлення: у методі `Update` перевіряється, чи є шайба (`puck`) і чи не активна (`puckActive = false`). Якщо так, бот обчислює напрямок до шайби та задає швидкість руху в цьому напрямку.

Вдарити шайбу: якщо бот знаходиться в межах відстані `hitDistance` до шайби, викликається метод `HitPuck`, який обчислює напрямок до воріт та застосовує силу на шайбу в цьому напрямку. Після удару шайба стає неактивною (`puckActive = false`).

Лістинг 2.1 – Скрипт «BotController»

```
using UnityEngine;

public class BotController : MonoBehaviour
{
    [SerializeField] private float moveSpeed = 5f;
    [SerializeField] private float hitDistance = 2f;
    [SerializeField] private float hitForce = 5f;
    [SerializeField] private Transform puck;
    [SerializeField] private Transform net;
    [SerializeField] BotController[] botControllers;

    private Rigidbody2D rb;
    bool puckActive = true;

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        Invoke("StartMovement", Random.Range(2f, 4f));
    }
    void StartMovement()
    {
        puckActive = false;
        moveSpeed = Random.Range(3f, 4f);
        hitDistance = Random.Range(0.5f, 1.5f);
        hitForce = Random.Range(2f, 5f);
    }
    void Update()
    {
        if (puck != null && !puckActive)
        {
            Vector2 direction = (puck.position -
transform.position).normalized;

            rb.velocity = direction * moveSpeed;
```

```

        float distanceToPuck =
Vector2.Distance(transform.position, puck.position);

        if (distanceToPuck <= hitDistance)
        {
            HitPuck(direction);
        }
    }

void HitPuck(Vector2 direction)
{
    Vector3 gatePosition = net.transform.position;
    Vector3 directionToGate = (gatePosition -
transform.position).normalized;
    puck.GetComponent<Rigidbody2D>().AddForce(directionToGate *
hitForce, ForceMode2D.Impulse);
    puckActive = false;
}
}

```

У лістингу 2.2 здійснюється керування інтерфейсом користувача в грі, зокрема обробляє натискання клавіш для перемикання між різними зображеннями, перезапуску сцени, переходу на головне меню та закриття панелі завершення гри.

Ініціалізація: оголошення приватних змінних для зберігання посилань на об'єкти інтерфейсу (image1, image2, panelFinish, punk) та на канву (canvas).

Оновлення: в методі Update відслідковуються натискання різних клавіш:

Вверх (UpArrow або W): при натисканні клавіши вгору або W, image1 стає активним (SetActive(true)), а image2 - неактивним (SetActive(false)).

Вниз (DownArrow або S): якщо натиснуто клавішу вниз або S, image1 стає неактивним (SetActive(false)), а image2 - активним (SetActive(true)).

Введення (Return): у разі натиснутої клавіши Enter, перевіряється, яке зображення активне: Якщо активне image1, перезавантажується поточна сцена (SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex)). Якщо активне image2, завантажується сцена з індексом 0 (SceneManager.LoadScene(0)), зазвичай це головне меню.

Клавіша Escape: якщо натиснуто клавішу Escape, гра продовжується (Time.timeScale = 1), канва відправляється на задній план (canvas.sortingOrder = 0), і панель завершення гри стає неактивною (panelFinish.SetActive(false)).

Таким чином, цей скрипт забезпечує обробку вводу користувача для перемикання між зображеннями, перезавантаження гри, переходу в головне меню та закриття панелі завершення гри.

Лістинг 2.2 – Скрипт «GameInterface»

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

public class GameInterface : MonoBehaviour
{
    [SerializeField] private GameObject image1, image2,
    panelFinish, punk;
    [SerializeField] private Canvas canvas;
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.UpArrow) ||
        Input.GetKeyDown(KeyCode.W))
        {
            image1.SetActive(true);
            image2.SetActive(false);
        }

        if (Input.GetKeyDown(KeyCode.DownArrow) ||
        Input.GetKeyDown(KeyCode.S))
        {
            image1.SetActive(false);
            image2.SetActive(true);
        }
        if (Input.GetKeyDown(KeyCode.Return))
        {
            if (image1.activeSelf)
            {
                SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
            }
            else if (image2.activeSelf)
            {
                SceneManager.LoadScene(0);
            }
        }
    }
}
```

```

        }
    }
    if (Input.GetKeyDown(KeyCode.Escape))
    {
        Time.timeScale = 1;
        canvas.sortingOrder = 0;
        panelFinish.SetActive(false);
    }
}
}

```

У лістингу 2.3 відтворюється звуковий ефект.

Ініціалізація: оголошується статична змінна `audioSource`, яка посилається на компонент `AudioSource`. Оголошується статична змінна `audioClipName`, яка містить назву звукового кліпу (за замовчуванням "Hit").

У методі `Start` компонент `AudioSource` отримується з об'єкта, до якого прикріплений цей скрипт, і зберігається в `audioSource`.

Метод `AudioClipOn`: завантажується звуковий кліп з папки `Resources` за назвою `audioClipName`. Завантажений кліп встановлюється як поточний кліп для `audioSource`. Відтворюється звуковий кліп за допомогою `audioSource.Play()`.

Метод `Update`: перевіряється значення параметра "Effects" у `PlayerPrefs`. Якщо значення "Effects" дорівнює 1, звук вимикається (`audioSource.mute = true`). Якщо значення "Effects" не дорівнює 1, звук увімкнено (`audioSource.mute = false`).

Таким чином, цей скрипт завантажує та відтворює звуковий ефект при виклику методу `AudioClipOn`, а також вмикає або вимикає звук на основі значення параметра "Effects" у `PlayerPrefs`.

Лістинг 2.3 – Скрипт «HitSound»

```

using UnityEngine;

public class HitSound : MonoBehaviour
{
    public static AudioSource audioSource;
    public static string audioClipName = "Hit";

    private void Start()
    {

```

```

        audioSource = GetComponent<AudioSource>();
    }

    public static void AudioClipOn()
    {
        AudioClip clip = Resources.Load<AudioClip>(audioClipName);
        audioSource.clip = clip;

        audioSource.Play();
    }

    private void Update()
    {
        if (PlayerPrefs.GetInt("Effects") == 1)
        {
            audioSource.mute = true;
        }
        else
        {
            audioSource.mute = false;
        }
    }
}

```

У лістингу 2.4 здійснюється керування рухом об'єкта між двома точками у Unity.

Ініціалізація: оголошуються змінні `point1` та `point2`, які посилаються на об'єкти типу `Transform` і представляють дві точки. Оголошується змінна `moveSpeed`, яка визначає швидкість руху об'єкта. Оголошується змінна `targetPoint`, яка посилається на цільову точку, до якої буде рухатись об'єкт.

У методі `Start` встановлюється початкова цільова точка (`targetPoint`) на `point1`.

Метод `Update`: об'єкт переміщується до поточної цільової точки (`targetPoint`) зі швидкістю `moveSpeed`, використовуючи метод `Vector2.MoveTowards`. Якщо об'єкт майже досяг цільової точки (відстань менше ніж 0.01 одиниці), перевіряється поточна цільова точка. Якщо поточна цільова точка — це `point1`, цільова точка змінюється на `point2`. Якщо поточна цільова точка — це `point2`, цільова точка змінюється на `point1`.

Таким чином, цей скрипт забезпечує безперервний рух об'єкта між двома точками зі заданою швидкістю, змінюючи напрямок руху, коли об'єкт досягає однієї з точок.

Лістинг 2.4 – Скрипт «KeeperMovement»

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

public class KeeperMovement : MonoBehaviour
{
    [SerializeField] private Transform point1;
    [SerializeField] private Transform point2;
    [SerializeField] private float moveSpeed = 5f;
    private Transform targetPoint;

    void Start()
    {
        targetPoint = point1;
    }

    void Update()
    {
        transform.position = Vector2.MoveTowards(transform.position,
        targetPoint.position, moveSpeed * Time.deltaTime);
        if (Vector2.Distance(transform.position,
        targetPoint.position) < 0.01f)
        {
            if (targetPoint == point1)
            {
                targetPoint = point2;
            }
            else
            {
                targetPoint = point1;
            }
        }
    }
}
```

У лістингу 2.5 здійснюється керування меню гри.

Ініціалізація: оголошуються змінні `image1`, `image2`, `image3`, `panelSetting`, `panelController`, які посилаються на відповідні об'єкти типу `GameObject`. Оголошуються змінні `audioSound` і `audioEffects`, які посилаються на компоненти

AudioSource. Оголошується змінна pos, яка відповідає за позицію вибраного елемента в меню (початкове значення 1).

Метод Start: встановлюється початкова позиція pos на 1. Перевіряється значення параметра "Sound" у PlayerPrefs.

Якщо значення дорівнює 1, звук вимикається (audioSound.mute = true), інакше звук увімкнений та починає відтворюватись (audioSound.Play()). Перевіряється значення параметра "Effects" у PlayerPrefs.

Якщо значення дорівнює 1, звукові ефекти вимикаються (audioEffects.mute = true), інакше звукові ефекти увімкнені.

Метод: ControlButton перевіряє натискання клавіш для керування меню. Якщо натиснута клавіша UpArrow або W і pos більше 1, зменшується значення pos. Якщо натиснута клавіша DownArrow або S і pos менше 3, збільшується значення pos.

Метод Update: викликається метод ControlButton для обробки вводу користувача. Активуються та деактивуються відповідні зображення в залежності від значення pos.

Якщо pos дорівнює 1, активується image1, а image2 та image3 деактивуються.

Якщо pos дорівнює 2, активується image2, а image1 та image3 деактивуються.

Якщо pos дорівнює 3, активується image3, а image1 та image2 деактивуються.

Якщо натиснута клавіша Return (Enter): Якщо панель налаштувань (panelSetting) не активна:

Якщо активна image1, завантажується сцена з індексом 1. Якщо активна image2, активується панель налаштувань (panelSetting). Якщо активна image3, активується панель контролю (panelController). Якщо натиснута клавіша Escape: Деактивуються панелі налаштувань (panelSetting) і контролю (panelController).

Таким чином, цей скрипт забезпечує навігацію в меню гри, керування звуками та відображення відповідних панелей налаштувань та контролю.

Лістинг 2.5 – Скрипт «Menu»

```

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

public class Menu : MonoBehaviour
{
    [SerializeField] private GameObject image1;
    [SerializeField] private GameObject image2;
    [SerializeField] private GameObject image3;
    [SerializeField] private GameObject panelSetting;
    [SerializeField] private GameObject panelController;
    [SerializeField] private AudioSource audioSound, audioEffects;

    public int pos = 1;
    private void Start()
    {
        pos = 1;

        if (PlayerPrefs.GetInt("Sound") == 1)
        {
            audioSound.mute = true;
        }
        else
        {
            audioSound.mute = false;
            audioSound.Play();
        }
        if (PlayerPrefs.GetInt("Effects") == 1)
        {
            audioEffects.mute = true;
        }
        else
        {
            audioEffects.mute = false;
        }
    }

    public void ControlButton()
    {
        if (Input.GetKeyDown(KeyCode.UpArrow) ||
            Input.GetKeyDown(KeyCode.W))
    
```

```

    {
        if (pos > 1)
        {
            pos--;
        }
    }

    if (Input.GetKeyDown(KeyCode.DownArrow) ||
Input.GetKeyDown(KeyCode.S))
    {
        if (pos < 3)
        {
            pos++;
        }
    }
}

void Update()
{
    ControlButton();

    if (pos == 1)
    {
        image1.SetActive(true);
        image2.SetActive(false);
        image3.SetActive(false);
    }
    else if (pos == 2)
    {
        image1.SetActive(false);
        image2.SetActive(true);
        image3.SetActive(false);
    }
    else if (pos == 3)
    {
        image1.SetActive(false);
        image2.SetActive(false);
        image3.SetActive(true);
    }

    if (Input.GetKeyDown(KeyCode.Return))
    {
        if (!panelSetting.activeSelf)
        {

```

```

        if (image1.activeSelf)
        {
            SceneManager.LoadScene(1);
        }
        else if (image2.activeSelf)
        {
            panelSetting.SetActive(true);
        }
        else if (image3.activeSelf)
        {
            panelController.SetActive(true);
        }
    }
}
if (Input.GetKeyDown(KeyCode.Escape)) {
    panelSetting.SetActive(false);
    panelController.SetActive(false);
}
}
}

```

Скрипт за лістингом 2.6 керує поведінкою гравця у грі Unity.

Ініціалізація: оголошуються змінні `speed` (швидкість руху гравця) та `hitForce` (сила удару) з початковими значеннями. Оголошується змінна `rb`, яка буде посиленням на компонент `Rigidbody2D` гравця. Оголошується змінна `push`, яка посиляється на об'єкт шайби. Оголошується змінна `transformChild`, яка посиляється на дочірній об'єкт трансформації (його використання не відображене у скрипті).

Метод `Start`: отримується компонент `Rigidbody2D` для гравця та зберігається в змінну `rb`.

Метод `FixedUpdate`: зчитуються горизонтальний (`moveHorizontal`) та вертикальний (`moveVertical`) вводи користувача. На основі цих значень створюється вектор руху `movement`. Швидкість руху гравця встановлюється за допомогою множення вектора руху на швидкість (`speed`). Перевіряється напрямок руху для відображення спрайту гравця:

Якщо гравець рухається вліво (`moveHorizontal < 0`), спрайт гравця відображається горизонтально (`flipX = true`).

Якщо гравець рухається вправо, спрайт гравця не відображається горизонтально (`flipX = false`).

Якщо натиснута клавіша `Space` і шайба прикріплена до гравця: Знімається флаг `isKinematic` для фізики шайби. Додається сила до шайби на основі швидкості гравця та сили удару (`hitForce`). Шайба відкріплюється від гравця. Викликається метод `AudioClipOn` для відтворення звукового ефекту удару (метод `HitSound.AudioClipOn()` не відображений у скрипті, передбачається, що він існує).

Метод `OnCollisionEnter2D`: якщо гравець стикається з іншим гравцем (об'єкт з тегом `"Player"`), шайба прикріплюється до гравця. Встановлюється флаг `isKinematic` для фізики шайби.

Таким чином, цей скрипт забезпечує керування рухом гравця, обробку удару шайби та прикріплення шайби до гравця при зіткненні з іншим гравцем.

Лістинг 2.6 – Скрипт «PlayerController»

```
using UnityEngine;

public class PlayerController : MonoBehaviour
{
    [SerializeField] private float speed = 5f;
    [SerializeField] private float hitForce = 10f;
    private Rigidbody2D rb;
    [SerializeField] private GameObject puck;
    [SerializeField] private Transform transformChild;

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    void FixedUpdate()
    {
        float moveHorizontal = Input.GetAxis("Horizontal");
        float moveVertical = Input.GetAxis("Vertical");
```

```

        Vector2 movement = new Vector2(moveHorizontal,
moveVertical);
        rb.velocity = movement * speed;

        if (moveHorizontal < 0)
        {
            GetComponent<SpriteRenderer>().flipX = true;
        }
        else
        {
            GetComponent<SpriteRenderer>().flipX = false;
        }

        if (Input.GetKey(KeyCode.Space))
        {
            if (puck.transform.parent != null)
            {
                puck.GetComponent<Rigidbody2D>().isKinematic =
false;

                Vector2 playerVelocity = rb.velocity;

                puck.GetComponent<Rigidbody2D>().AddForce(playerVelocity * hitForce,
ForceMode2D.Impulse);
                puck.transform.SetParent(null);

                HitSound.AudioClipOn();
            }
        }

    }

    private void OnCollisionEnter2D(Collision2D other)
    {
        if (other.gameObject.CompareTag("Player"))
        {
            puck.transform.SetParent(transform);
            puck.GetComponent<Rigidbody2D>().isKinematic = true;
        }
    }
}

```

Скрипт за лістингом 2.7 керує перемиканням між різними гравцями.

Ініціалізація: оголошується змінна `switchKey`, яка визначає клавішу для перемикавання гравців (за замовчуванням клавіша E). Оголошуються масиви `players` та `playersChild`, які містять посилання на контролери гравців та їх дочірні об'єкти відповідно. Оголошується змінна `activePlayerIndex`, яка відстежує індекс активного гравця (початкове значення 0). Оголошуються змінні `activeColor` та `defaultColor`, які визначають кольори для активного та неактивних гравців відповідно.

Метод `Start`: викликається метод `SetPlayerColors`, щоб встановити початкові кольори гравців.

Метод `Update`: перевіряється, чи натиснута клавіша перемикавання (`switchKey`).

Якщо натиснута, викликається метод `SwitchPlayer`. Метод `SwitchPlayer`: Вимикає контролер для поточного активного гравця (`players[activePlayerIndex].enabled = false`). Збільшує індекс активного гравця (`activePlayerIndex`).

Якщо індекс перевищує кількість гравців, він скидається до 0 (`activePlayerIndex = 0`). Вмикає контролер для нового активного гравця (`players[activePlayerIndex].enabled = true`). Викликається метод `SetPlayerColors` для оновлення кольорів гравців.

Метод `SetPlayerColors`: проходить через всіх гравців і встановлює відповідний колір. Якщо гравець активний (індекс відповідає `activePlayerIndex`), встановлюється `activeColor`. Якщо гравець неактивний, встановлюється `defaultColor`.

Метод `ChangePlayerColor`: змінює колір переданого об'єкта гравця. Отримує компонент `SpriteRenderer` для об'єкта. Якщо компонент існує, змінює його колір на заданий.

Таким чином, цей скрипт забезпечує перемикавання між кількома гравцями в грі за допомогою натискання клавіші та візуально відображає активного гравця, змінюючи його колір.

Лістинг 2.7 – Скрипт «PlayerSwitcher»

```
using UnityEngine;

public class PlayerSwitcher : MonoBehaviour
{
    [SerializeField] private KeyCode switchKey = KeyCode.E;
    [SerializeField] private PlayerController[] players;
    [SerializeField] private GameObject[] playersChild;

    private int activePlayerIndex = 0;

    [SerializeField] private Color activeColor;
    [SerializeField] private Color defaultColor;

    void Start()
    {
        SetPlayerColors();
    }

    void Update()
    {
        if (Input.GetKeyDown(switchKey))
        {
            SwitchPlayer();
        }
    }

    void SwitchPlayer()
    {
        players[activePlayerIndex].enabled = false;
        activePlayerIndex++;
        if (activePlayerIndex >= players.Length)
        {
            activePlayerIndex = 0;
        }

        players[activePlayerIndex].enabled = true;
        SetPlayerColors();
    }

    void SetPlayerColors()
    {
        for (int i = 0; i < players.Length; i++)
```

```

    {
        if (i == activePlayerIndex)
        {
            ChangePlayerColor(playersChild[i], activeColor);
        }
        else
        {
            ChangePlayerColor(playersChild[i], defaultColor);
        }
    }
}

void ChangePlayerColor(GameObject player, Color color)
{
    SpriteRenderer renderer =
player.GetComponent<SpriteRenderer>();
    if (renderer != null)
    {
        renderer.color = color;
    }
}
}

```

Скрипт за лістингом 2.8 керує поведінкою шайби.

Ініціалізація: оголошуються змінні `pointText` та `finalText`, які посилаються на текстові об'єкти для відображення кількості очок гравця і противника в режимі гри та на екрані завершення гри відповідно. Оголошуються змінні `playerPoint` та `enemyPoint`, які відстежують кількість очок гравця та противника відповідно. Оголошуються змінні `player` (посилання на об'єкт гравця), `players` (масив об'єктів гравців) та `pointSpawn` (масив точок для респауну гравців). Оголошується змінна `bounceForce`, яка визначає силу відбиття шайби.

Метод `Start`: отримується компонент `Rigidbody2D` для шайби та зберігається в змінну `rb`. Зчитуються значення кількості очок гравця і противника з `PlayerPrefs`. Встановлюється текст для відображення кількості очок гравця та противника.

Метод `OnCollisionEnter2D`: перевіряється колізія з межею гри. Обчислюється вектор відбиття шайби за допомогою методу `Vector2.Reflect`.

Встановлюється нова швидкість шайби, використовуючи отриманий вектор відбиття та задану силу відбиття.

Метод `OnTriggerEnter2D`: перевіряється, чи шайба зіштовхнулася з метою гравця або противника. Відповідно до тегу об'єкта (мета гравця чи противника), збільшується відповідна кількість очок. Оновлюється відображення кількості очок гравця та противника на екрані.

Метод `Respawn`: перебирається масив гравців. Шайба відкріплюється від батьківського об'єкта. Обнулюється швидкість кожного гравця. Кожен гравець переміщується до відповідної точки респауну з масиву `pointSpawn`.

Цей скрипт дозволяє керувати рухом та взаємодією шайби з межею гри та метами гравця і противника, а також оновлювати відображення кількості очок на екрані.

Лістинг 2.8 – Скрипт «Puck»

```
using TMPro;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.SceneManagement;

public class Puck : MonoBehaviour
{
    [SerializeField] TMP_Text pointText, finalText;
    int playerPoint = 0;
    int enemyPoint = 0;

    [SerializeField] Transform player;
    [SerializeField] GameObject[] players;
    [SerializeField] Transform[] pointSpawn;
    public float bounceForce = 10f;
    Rigidbody2D rb;
    private void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        playerPoint = PlayerPrefs.GetInt("PlayerPoint");
        enemyPoint = PlayerPrefs.GetInt("EnemyPoint");
        pointText.text = $"{playerPoint} : {enemyPoint}";
    }
}
```

```

void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.CompareTag("Border"))
    {
        Vector2 reflection = Vector2.Reflect(rb.velocity,
collision.contacts[0].normal).normalized;
        rb.velocity = reflection * bounceForce;
    }
}
private void OnTriggerEnter2D(Collider2D other)
{
    if (other.gameObject.CompareTag("HomeNet"))
    {
        enemyPoint++;
        pointText.text = $"{playerPoint} : {enemyPoint}";
        finalText.text = $"{playerPoint} : {enemyPoint}";
        PlayerPrefs.SetInt("PlayerPoint", enemyPoint);
        Respawn();
    }
    if (other.gameObject.CompareTag("HomeEnemy"))
    {
        playerPoint++;
        pointText.text = $"{playerPoint} : {enemyPoint}";
        finalText.text = $"{playerPoint} : {enemyPoint}";
        PlayerPrefs.SetInt("EnemyPoint", playerPoint);
        Respawn();
    }
}

void Respawn()
{
    for (int i = 0; i < players.Length; i++)
    {
        transform.SetParent(null);
        players[i].GetComponent<Rigidbody2D>().velocity =
Vector2.zero;
        players[i].transform.position = pointSpawn[i].position;
    }
}
}

```

Скрипт за лістингом 2.9 відповідає за слідування за цільовим об'єктом.

Ініціалізація: оголошуються змінні `target` (цільовий об'єкт), `followDistance` (відстань, на яку буде слідувати об'єкт) та `smoothSpeed` (швидкість плавного руху до цільового об'єкту).

Метод `FixedUpdate`: перевіряється, чи цільовий об'єкт існує (`target != null`). Обчислюється напрямок до цільового об'єкту як різниця між його позицією та позицією об'єкту, до якого цей скрипт прикріплений. Обчислюється бажана позиція для об'єкту, яка знаходиться на відстані `followDistance` від цільового об'єкту. Застосовується плавний перехід до бажаної позиції за допомогою методу `Vector3.Lerp`. Цей скрипт забезпечує плавне слідування об'єкту за цільовим об'єктом з заданою відстанню.

Лістинг 2.9 – Скрипт «PuckFollow»

```
using UnityEngine;

public class PuckFollow : MonoBehaviour
{
    [SerializeField] private Transform target;
    [SerializeField] private float followDistance = 2f;
    [SerializeField] private float smoothSpeed = 0.125f;

    void FixedUpdate()
    {
        if (target != null)
        {
            Vector3 directionToTarget = target.position -
transform.position;
            Vector3 desiredPosition = target.position -
directionToTarget.normalized * followDistance;
            transform.position = Vector3.Lerp(transform.position,
desiredPosition, smoothSpeed);
        }
    }
}
```

Скрипт за лістингом 2.10 керує налаштуваннями звуку в грі.

Ініціалізація: оголошуються змінні `image1` та `image2`, які посилаються на два об'єкти UI для відображення налаштувань звуку. Оголошуються змінні

audioSound та audioEffects, які посиляються на компоненти AudioSource для звуку та звукових ефектів відповідно. Оголошуються змінні textSound та textEffect, які посиляються на тексти для відображення статусу звуку та звукових ефектів. Ініціалізуються змінні activeSound та activeEffects як false.

Метод Start: перевіряється, чи вимкнений звук та звукові ефекти відповідно до значень, збережених у PlayerPrefs. Відповідно до значень PlayerPrefs встановлюються тексти та флаги для активації/деактивації звуку та звукових ефектів. Залежно від стану, встановлюється відповідне значення mute для AudioSource.

Метод Update: перевіряється введення клавіш для переміщення вгору і вниз (UpArrow, W, DownArrow, S). При натисканні клавіш змінюються стан UI елементів та відтворюється звук. Перевіряється натискання клавіші Return. Якщо активний перший UI елемент: Змінюється стан звуку та зберігається в PlayerPrefs. Якщо активний другий UI елемент: Змінюється стан звукових ефектів та зберігається в PlayerPrefs.

Цей скрипт дозволяє гравцеві вмикати або вимикати звук та звукові ефекти та зберігати їх стан між сеансами гри.

Лістинг 2.10 – Скрипт «Settings»

```
using TMPro;
using UnityEngine;

public class Settings : MonoBehaviour
{
    [SerializeField] private GameObject image1;
    [SerializeField] private GameObject image2;
    [SerializeField] public AudioSource audioSound, audioEffects;
    [SerializeField] private TMP_Text textSound, textEffect;
    bool activeSound, activeEffects = false;
    private void Start()
    {
        if (PlayerPrefs.GetInt("Sound") == 1)
        {
            textSound.text = "OFF";
            activeSound = true;
        }
    }
}
```

```

        audioSound.mute = true;
    }
    else
    {
        textSound.text = "ON";
        activeSound = false;
        audioSound.mute = false;
    }
    if (PlayerPrefs.GetInt("Effects") == 1)
    {
        textEffect.text = "OFF";
        activeEffects = true;
        audioEffects.mute = true;
    }
    else
    {
        textEffect.text = "ON";
        activeEffects = false;
        audioEffects.mute = false;
    }
}
void Update()
{
    if (Input.GetKeyDown(KeyCode.UpArrow) ||
Input.GetKeyDown(KeyCode.W))
    {
        image1.SetActive(true);
        image2.SetActive(false);
        audioSound.Play();
    }

    if (Input.GetKeyDown(KeyCode.DownArrow) ||
Input.GetKeyDown(KeyCode.S))
    {
        image1.SetActive(false);
        image2.SetActive(true);
        audioSound.Play();
    }
    if (Input.GetKeyDown(KeyCode.Return))
    {
        if (image1.activeSelf)
        {
            if (!activeSound)
            {

```

```

        textSound.text = "OFF";
        activeSound = true;
        audioSound.mute = true;
        PlayerPrefs.SetInt("Sound", 1);
    }
    else
    {
        textSound.text = "ON";
        activeSound = false;
        audioSound.mute = false;
        PlayerPrefs.SetInt("Sound", 0);
    }
}
else if (image2.activeSelf)
{
    if (!activeEffects)
    {
        textEffect.text = "OFF";
        activeEffects = true;
        audioEffects.mute = true;
        PlayerPrefs.SetInt("Effects", 1);
    }
    else
    {
        textEffect.text = "ON";
        activeEffects = false;
        audioEffects.mute = false;
        PlayerPrefs.SetInt("Effects", 0);
    }
}
}
}
}

```

Скрипт за лістингом 2.11 відповідає за таймер у грі Unity.

Ініціалізація: оголошуються змінні `totalTime` (загальний час таймера), `accelerationInterval` (інтервал для прискорення таймера) та `accelerationAmount` (величина прискорення). Оголошуються змінні `currentTime` (поточний час на таймері) та `lastAccelerationTime` (час останнього прискорення таймера). Оголошуються посилання на тексти (`timerText`) та канвас (`canvas`), а також

посилання на панель (finalPanel), що відображається в кінці гри. Встановлюється початкове значення PlayerPrefs для кількості очок гравця та противника.

Метод Start: встановлюється початкове значення для часу таймера (currentTime) та часу останнього прискорення (lastAccelerationTime).

Метод Update: перевіряється натискання клавіші Escape для паузи гри. Зменшується поточний час таймера.

Якщо час таймера закінчився, гра зупиняється, очки обнуляються, відображається панель завершення гри.

Якщо пройшов достатній час з моменту останнього прискорення, викликається метод AccelerateTimer() для прискорення таймера. Оновлюється відображення таймера.

Метод UpdateTimerDisplay: обчислюється та відображається час у форматі хвилин та секунд.

Метод AccelerateTimer: зменшується поточний час таймера на величину прискорення. Оновлюється час останнього прискорення.

Метод Continue: продовжується гра (відновлюється час), сховується панель завершення гри.

Цей скрипт дозволяє відслідковувати час у грі, прискорювати його певним інтервалом та управляти паузою гри.

Лістинг 2.11 – Скрипт «Timer»

```
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class Timer : MonoBehaviour
{
    [SerializeField] private float totalTime = 10f;
    [SerializeField] private float accelerationInterval = 1f;
    [SerializeField] private float accelerationAmount = 1f;
    private float currentTime;
    private float lastAccelerationTime;
    [SerializeField] private TMP_Text timerText;
    [SerializeField] private Canvas canvas;
```

```

[SerializeField] private GameObject finalPanel;
bool pause = false;
private void Awake()
{
    PlayerPrefs.SetInt("EnemyPoint", 0);
    PlayerPrefs.SetInt("PlayerPoint", 0);
}
void Start()
{
    Time.timeScale = 1;
    currentTime = totalTime;
    lastAccelerationTime = Time.time;
}

void Update()
{
    if (Input.GetKeyDown(KeyCode.Escape))
    {
        Time.timeScale = 0;
        canvas.sortingOrder = 4;
        finalPanel.SetActive(true);
    }

    currentTime -= Time.deltaTime;

    if (currentTime <= 0)
    {
        currentTime = 0;
        Time.timeScale = 0;
        PlayerPrefs.SetInt("EnemyPoint", 0);
        PlayerPrefs.SetInt("PlayerPoint", 0);
        canvas.sortingOrder = 4;
        finalPanel.SetActive(true);
    }

    if (Time.time - lastAccelerationTime >=
accelerationInterval)
    {
        AccelerateTimer();
    }

    UpdateTimerDisplay();
}

```

```

    }

    void UpdateTimerDisplay()
    {
        int minutes = Mathf.FloorToInt(currentTime / 60);
        int seconds = Mathf.FloorToInt(currentTime % 60);
        string timeString = string.Format("{0:00}:{1:00}", minutes,
seconds);
        timerText.text = timeString;
    }

    void AccelerateTimer()
    {
        currentTime -= accelerationAmount;
        lastAccelerationTime = Time.time;
    }

    public void Continue()
    {
        Time.timeScale = 1;
        canvas.sortingOrder = 0;
        finalPanel.SetActive(false);
        pause = false;
    }
}

```

2.5 Тестування

Тестування є невід'ємною частиною розробки гри, що включає перевірку функціональності, продуктивності, сумісності та зручності використання. Тестувальники перевіряють гру на наявність помилок, багів та інших недоліків, які можуть вплинути на геймплей. Важливо проводити тестування на різних пристроях та браузерах, щоб забезпечити стабільну роботу гри для всіх користувачів.

Основні етапи тестування включають:

- функціональне тестування: перевіряється, чи працюють всі функції гри відповідно до вимог дизайну. Виявляються та виправляються помилки, що впливають на геймплей. Це включає перевірку всіх ігрових механік, систем досягнень, збереження прогресу та інших важливих аспектів гри. Наприклад, крім основних механік, тестувальники перевіряють різні варіанти взаємодії гравця з грою, а також взаємодію різних елементів гри між собою;

- тестування продуктивності: перевіряється швидкодія гри, її стабільність та використання ресурсів. Це включає тестування гри при різних навантаженнях, перевірку використання пам'яті, CPU та GPU, а також тестування на різних пристроях з різними технічними характеристиками. Крім того, важливо визначити оптимальні налаштування для різних типів пристроїв;

- сумісність тестування: перевіряється коректна робота гри на різних пристроях та операційних системах. Важливо, щоб гра працювала однаково добре як на десктопах, так і на мобільних пристроях, а також у різних браузерах (Chrome, Firefox, Edge та інші). Під час цього етапу також враховуються специфіка мобільних пристроїв та різниця в операційних системах;

- юзабіліті тестування: оцінюється зручність використання гри з точки зору користувача. Проводяться опитування та збираються відгуки від гравців, щоб виявити можливі проблеми та покращити ігровий досвід. Під час цього етапу також важливо оцінити ергономіку контролерів та адаптивний дизайн інтерфейсу;

- регресійне тестування: після виправлення помилок проводиться повторне тестування для перевірки, що зміни не викликали нових проблем. Це дозволяє забезпечити стабільність гри та уникнути повторного виникнення вже виправлених помилок.

Тестування є критичним етапом, що дозволяє виявити та виправити помилки до релізу гри. Воно забезпечує стабільну та якісну роботу продукту, що сприяє позитивному досвіду гравців та успіху гри на ринку.

2.6 Реліз

Реліз гри є фінальним етапом розробки, що включає підготовку продукту до випуску та його представлення користувачам.

Цей етап включає кілька важливих кроків:

- підготовка до релізу: на цьому етапі завершується тестування та виправлення останніх помилок. Гра проходить фінальне тестування, щоб переконатися у її стабільності та готовності до випуску. Підготовка також включає створення маркетингових матеріалів, таких як трейлери, скріншоти та опис гри;

- створення релізних збірок: гра компілюється у фінальну збірку, яка буде розповсюджуватись серед користувачів. Для браузерної гри це може включати оптимізацію коду та ресурсів для швидкого завантаження та коректної роботи у веб-браузерах;

- розповсюдження: гра завантажується на платформи розповсюдження, такі як Steam, Epic Games Store, App Store, Google Play або спеціалізовані ігрові портали. Для браузерної гри це може включати публікацію на власному веб-сайті або платформах, що підтримують WebGL;

- маркетинг та промоція: важливо провести рекламну кампанію для залучення уваги до нової гри. Це може включати оголошення в соціальних мережах, прес-релізи, рекламні статті, відеоогляди та інші види промоції. Співпраця з ігровими блогерами та стримерами також може суттєво підвищити впізнаваність гри;

- підтримка після релізу: після випуску гри важливо підтримувати зв'язок з гравцями, оперативно виправляти виявлені помилки та оновлювати контент. Підтримка також включає відповіді на питання користувачів, надання технічної допомоги та випуск оновлень, що покращують гру.

Реліз гри є кульмінацією багатомісячної або навіть багаторічної праці команди розробників. Успішний реліз залежить від ретельної підготовки, якісного тестування та ефективної маркетингової стратегії.

2.7 Післявипускний цикл гри: забезпечення постійного успіху та задоволення гравців

Однією з перших та найважливіших дій після випуску гри є активний моніторинг реакції користувачів. Розробники аналізують відгуки, коментарі та рецензії гравців на форумах, соціальних мережах та в ігрових магазинах. Такий моніторинг дозволяє виявляти та вирішувати проблеми швидше, що сприяє підвищенню задоволення гравців та покращенню репутації гри.

Розробники продовжують підтримувати гру, випускаючи оновлення та патчі для виправлення помилок, оптимізації гри та додавання нового контенту. Постійні оновлення додають до гри нових можливостей та тримають зацікавленість гравців, а також вказують на те, що розробники продовжують дбати про свій продукт та його спільноту.

Після релізу гри маркетингові зусилля не припиняються. Розробники продовжують активно просувати гру через різноманітні канали, включаючи рекламу в інтернеті, соціальні мережі, співпрацю з впливовими людьми та організацію рекламних кампаній. Ефективна реклама допомагає привернути нових гравців та підтримувати інтерес до гри серед існуючих користувачів, що сприяє її популярності та успіху на ринку.

Для збільшення тривалості гри та привертання нових гравців розробники можуть працювати над створенням додаткового контенту. Додавання нового контенту регулярно дозволяє зберігати інтерес гравців та розширювати можливості гри, що робить її більш привабливою для широкої аудиторії.

Команда підтримки гравців продовжує працювати над вирішенням технічних проблем, відповідає на запитання гравців та надає технічну підтримку через різні канали зв'язку. Підтримка користувачів є важливою складовою успіху гри, оскільки вона дозволяє гравцям вирішувати будь-які труднощі, що вони можуть зустрітися під час гри, та створює позитивний досвід взаємодії з продуктом.

Розробники аналізують дані про продажі, активність гравців та інші метрики успіху для оцінки ефективності гри та прийняття стратегічних рішень щодо її подальшого розвитку. Вони вивчають показники, такі як кількість проданих копій гри, середній час гри кожного користувача, частота використання різних функцій тощо. Це допомагає розробникам зрозуміти що потребує покращення та визначити пріоритети для майбутніх оновлень.

Розробники активно взаємодіють з гравцями, організовуючи анкети, опитування та відкриті дні, щоб збирати зворотній зв'язок та ідеї для майбутніх оновлень та доповнень. Взаємодія зі спільнотою дозволяє розробникам краще зрозуміти потреби та побажання гравців, а також відчувати їх настрій щодо гри. Це робить розробку більш прозорою та допомагає створити продукт, який дійсно відповідає очікуванням гравців.

Деякі розробники надають підтримку моддингу, що дозволяє гравцям створювати власний контент для гри, що сприяє подальшому розвитку гри та залученню нових гравців. Створення спільноти розробників дозволяє гравцям обмінюватися ідеями, досвідом та створювати нові можливості для гри, що робить її більш привабливою для гравців, які шукають глибше спілкування та взаємодію.

Розробники продовжують працювати над поліпшенням гри через виправлення помилок, оптимізацію та додавання нового контенту для забезпечення позитивного ігрового досвіду. Постійне вдосконалення є ключовою складовою успіху гри в довгостроковій перспективі, оскільки воно дозволяє грі залишатися актуальною та привабливою для гравців у змінному і конкурентному ігровому середовищі.

ВИСНОВКИ

У ході виконання дипломної роботи бакалавра була сформульована задача розробки гри для браузерів з підтримкою WebGL. Були розглянуті основні етапи розробки браузерної гри за допомогою фреймворку Unity та технології WebGL.

У першому розділі була детально описана предметна область розробки браузерних ігор на ігровому двигуні Unity. Відзначено, що Unity - це потужне інтегроване середовище розробки, яке забезпечує розробникам ігор зручний та ефективний інструментарій для створення якісних ігрових додатків. Розглянуті основні особливості та можливості Unity, які роблять його популярним серед розробників браузерних ігор. Зазначено, що даний розділ надає чітке уявлення про те, як використовувати Unity для створення ігор для платформи WebGL, враховуючи усі особливості та можливості цього інструменту.

У другому розділі описані та виконані усі необхідні етапи розробки. Складання дизайн-документа є важливим етапом, оскільки він дає чітке бачення гри. Відсутність або недотриманість єдиного стилю візуальних або/та звукових елементів може впливати на загальні враження гравця стосовно гри. Ігровий процес - це основа будь-якої гри, він визначає взаємодію гравця з ігровим світом. Без ігрового процесу гра перетворюється на просту симуляцію або набір статичних сцен, які не залучають гравця та не дають йому можливості впливати на те, що відбувається. Від складності та комплексності гри будуть також залежати враження гравця про гру.

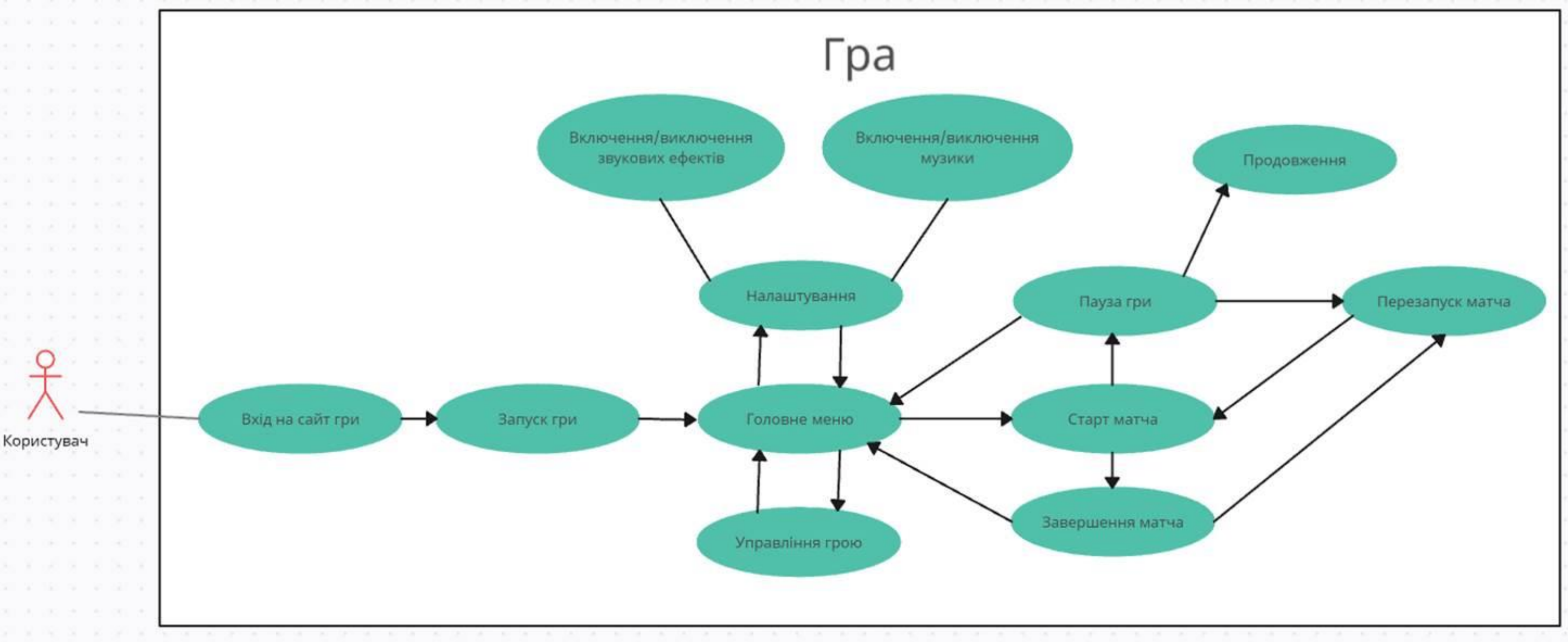
Таким чином, всі поставлені задачі дипломної роботи виконані, що свідчить про досягнення поставленої мети.

Надалі планується продовжити розробку даної гри, але вже для операційної системи «Windows» за документом, що складаний у пункті 2.1.2.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Adams E. Fundamentals of Game Design. New Riders, 2013. 576 с.
2. Burgun K. Game Design Theory: A New Philosophy for Understanding Games. CRC Press, 2012. 188 с.
3. Gregory J. Game Engine Architecture. CRC Press, 2018. 1200 с.
4. Hocking J. Unity in Action: Multiplatform game development in C#. Manning, 2018. 400 p.
5. Juul J. A Casual Revolution: Reinventing Video Games and Their Players. MIT Press, 2012. 264 с.
6. Koster R. Theory of Fun for Game Design. O'Reilly Media, Inc., 2013. 300 с.
7. Mike G. Unity 2022 Game Development in 24 Hours, Sams Teach Yourself. 4-те вид. Pearson Education, 2022. 464 с.
8. Rogers S. Level Up!: The Guide to Great Video Game Design. 2-ге вид. John Wiley & Sons Inc, 2014. 535 с.
9. Short T., Adams T. Procedural Generation in Game Design. CRC Press, 2017. 336 с.
10. Unity User Manual 2022.3 URL:
<https://docs.unity3d.com/manual/index.html>
11. Chocolate chip by uncle morris (Music Collection ,free download). URL:
<https://youtu.be/QbhbdFse4DQ?si=ZrcXczJfxrsD9R3N>
12. Ice Hockey asset-pack. URL: [https://chasersgaming.itch.io/asset-pack-ice-hockey-nes](https://chasersgaming itch io/asset-pack-ice-hockey-nes)
13. Kadith's Icons. URL: [https://kadith.itch.io/kadiths-free-icons](https://kadith.itch io/kadiths-free-icons)
14. Sound effects Pack 2. URL: [https://phoenix1291.itch.io/sound-effects-pack-2](https://phoenix1291.itch io/sound-effects-pack-2)
15. Браузерна гра «Ice Blitz». URL: [https://akutnyak.itch.io/iceblitz](https://akutnyak.itch io/iceblitz)

Діаграма прецедентів



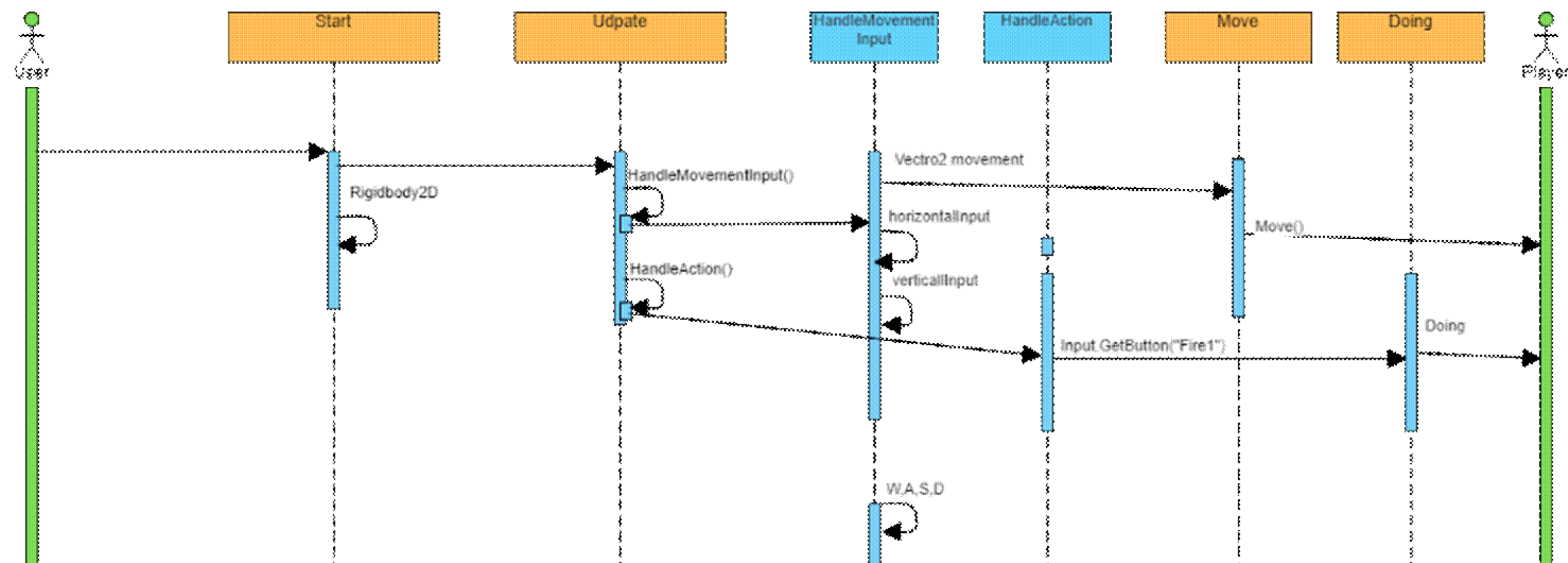
					13.02070849.00064 ПЛ1				
					Розробка аркадної спортивної 2D гри «Ice Blitz» на базі фреймворку Unity	Лит.		Маса	Масшт.
Зам.	Лист.	№ докум.	Підп.	Дата					
Розроб.		Кутняк А.С.	<i>Кутняк</i>	18.06.2024					
Перев.		Кудерметов Р.К.	<i>Кудерметов</i>						
Т.контр.					Діаграма прецедентів	Лист 1		Листів 3	
Н.контр.		Польська О.В.	<i>Польська</i>			НУ «Запорізька політехніка»			
Затв.		Кудерметов Р.К.	<i>Кудерметов</i>			КНТз-511сп			

Сайт гри «Ice Blitz»



					13.02070849.00064 ПЛ2				
					Розробка аркадної спортивної 2D гри «Ice Blitz» на базі фреймворку Unity	Лит.		Маса	Масшт.
Зам.	Лист.	№ докум.	Підп.	Дата					
Розроб.		Кутняк А.С.	<i>Кутняк</i>	18.06.2024					
Перев.		Кудерметов Р.К.	<i>Кудерметов</i>						
Т.контр.									
					Сайт гри «Ice Blitz»	Лист 2		Листів 3	
Н.контр.		Польська О.В.	<i>Польська</i>			НУ «Запорізька політехніка»			
Затв.		Кудерметов Р.К.	<i>Кудерметов</i>			КНТЗ-511сп			

Діаграма послідовності



					13.02070849.00064 ПЛЗ						
					Розробка аркадної спортивної 2D гри «Ice Blitz» на базі фреймворку Unity						
Зам.	Лист.	№ докум.	Підп.	Дата							
Розроб.		Кутняк А.С.	Кутняк	18.06.2024							
Перев.		Кудерметов Р.К.	Кудерметов								
Т.контр.											
					Діаграма послідовності						
Н.контр.		Польська О.В.	Польська								
Затв.		Кудерметов Р.К.	Кудерметов								
					Лист 3			Листів 3			
					НУ «Запорізька політехніка» КНТЗ-511сп						