РЕФЕРАТ

Пояснювальна записка: ___ с., ___ рис., ___ табл., __ дод., ___ джерел.

Об'єкт розробки: комп’ютерна гра.

Мета кваліфікаційної роботи: аналіз існуючих засобів розробки мультимедійних ігор та створення власної гри жанру раннер.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проведено аналіз предметної галузі, визначено актуальність завдання та призначення розробки, розроблена постановка завдання, задані вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі виконано аналіз існуючих рішень, обрано вибір платформи для розробки, виконано проектування і розробка програми, наведено опис алгоритму і структури функціонування програми, визначені вхідні і вихідні дані, наведені характеристики складу параметрів технічних засобів, описаний виклик та завантаження програми, описана робота програми.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Практичне значення роботи полягає в реалізації сучасними засобами цікавої комп’ютерної гри, що призначена для розважальних цілей.

Актуальність теми кваліфікаційної роботи визначається тим, що через те, що кожного дня ринок мобільних ігор дуже швидко змінюється та збільшується. аналіз сучасних засобів та окреслення сфер їх використання поліпшить якість та швидкість розробки ігрових проектів.

Список ключових слів: ГЕЙМЕР, ГРА, КОМП’ЮТЕР, ПЛАТФОРМА, ІНСТРУМЕНТАЛЬНІ ЗАСОБИ, ПРОЄКТУВАННЯ, ТЕСТУВАННЯ.

ABSTRACT

Explanatory note: ___ pp., ___ fig., ___ table, __ appendix, ___ sources.

Object of development: computer game.

The purpose of the qualification work: analysis of existing means of developing multimedia games and creating your own game of the runner genre.

The introduction considers the analysis and the current state of the problem, specifies the purpose of the qualification work and the field of its application, provides a justification for the relevance of the topic and clarifies the problem.

In the first section the analysis of the subject area is carried out, the urgency of the task and purpose of development is defined, the statement of the task is developed, requirements to software realization, technologies and software are set.

The second section analyzes existing solutions, selects a platform for development, performs design and development of the program, describes the algorithm and structure of the program, determines the input and output data, provides characteristics of the parameters of hardware, describes the call and download of the program, describes the program .

The economic section determines the complexity of the developed information system, calculates the cost of work to create a program and calculates the time for its creation.

The practical significance of the work lies in the realization by modern means of an interesting computer game designed for entertainment purposes.

The relevance of the topic of qualifying work is determined by the fact that every day the market for mobile games is changing and growing very quickly. analysis of modern tools and delineation of areas of their use will improve the quality and speed of development of game projects.

Keyword list: GAMER, GAME, COMPUTER, PLATFORM, INSTRUMENTS, DESIGN, TESTING.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОС – операційна система;

ПЗ – програмне забезпечення;

ПК – персональний комп’ютер$

UI – User Interface, інтерфейс користувача.

ЗМІСТ

ВСТУП

В сфері мультимедіа не останнє місце займає таке явище, як мультимедійні комп'ютерні ігри. Мультимедійні ігри - такі ігри, у яких гравець взаємодіє з віртуальним середовищем, побудованим комп'ютером. Стан віртуального середовища передається гравцеві за допомогою різних способів передачі інформації (аудіальний, візуальний, тактильний). Наразі всі комп'ютерні ігри відносяться до мультимедійних ігор. В такий тип ігор можна грати як в поодинці на локальному комп'ютері або приставці, так і з іншими гравцями через локальну або глобальну мережу.

Популярність комп‘ютерних ігор зростає з кожним роком, ніша ринки дозвілля постійно потребує поповнень новими ігровими тайтлами. Ігрова індустрія постійно розробляє нові інструментальні засоби для полегшення та пришвидшення процесу розробки ігор. Велика різноманітність, конкурентоздатність, специфічність використання, зміна актуальності використаних технологій роблять задачу вибору засобів розробки гри неоднозначною. Отже аналіз сучасних засобів та окреслення сфер їх використання є актуальною задачею, що поліпшить якість та швидкість розробки ігрових проектів, через те, що кожного дня ринок мобільних ігор дуже швидко змінюється та збільшується.

Метою кваліфікаційної роботи є аналіз існуючих засобів розробки мультимедійних ігор та створення власної гри жанру раннер.

Після проведення аналізу популярної гри «Angry Birds 2» було виявлено наступні недоліки: не завжди можна грати коли того хоче гравець, присутня реклама та мікротранзакції, ігрових рівнів кінцева кількість, неадаптованість гри під геймпад та вага у 371 МБ.

Було вирішено створити власну гру, яка б мала наступні виправлення згаданих вище недоліків: гравець може грати, коли захоче, відсутня реклама та мікротранзакції, відсутній екран завантаження, рейтинг гравців та досягнення

реалізовано за допомогою плагіну «Google Play Services», жанр гри – раннер, тобто гра є безкінечною, адаптовано під геймпад та має меншу вагу.

Вихідні дані до роботи:

1. Порівняння сучасних популярних інструментальних засобів: XNA, MonoGAMe, Phaser, Construct 2, pixi.js, Blender Game Engine, Easel.JS, Turbulenz, melonJS, PandaJS, Game Maker, Unity, Unreal Engine.
2. Визначення оптимальних засобів розробки комп'ютерних ігор.
3. Написання прототипу гри.

Для досягнення поставленої мети було обрано наступні програми: «Unity 3D», «Adobe Photoshop», «Magica Voxel 3D», «Visual Studio», додаток «Google Play Developer Console».

Для гри було створено два ігрових режими, кожен з яких доповнено унікальними механіками. Основна механіка обох режимів – вагонетка, яка рухається, і в яку необхідно влучити.

Практичне значення роботи полягає в реалізації сучасними засобами цікавої комп'ютерної гри, що призначена для розважальних цілей.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

1.1.1. Загальні положення з розробки комп’ютерних ігор

Актуальність цієї роботи визначається великою кількістю відеоігор на сьогоднішній день. Щодня мільйони людей грають, дивляться тематичні новини і спостерігають за грою інших людей. Ця сфера сьогодні є дуже вагомою і вона дуже швидко розвивається.

Історія виникнення комп’ютерних ігор бере свій початок у 1947 році. Саме тоді розвинення телебачення посприяло розвиненню відеоігор. Британський математик Алан Тьюрінг написав першу в світі теоретичну програму для гри у шахи, але, нажаль, тодішні комп’ютери не були здатні до її відтворення через свою невелику потужність.

Сьогодні сфера розробки відеоігор – “Gamedev” (походить від двох англійських слів – “Game” – гра та “Dev” тобто “Development” - розвиток) – є дуже поширеною. Також за весь час її існування з’явились певні правила, яких притримуються усі розробники. Найголовніше з них – ділення розробки на етапи.

Тож, розробка гри зазвичай ділиться на багато етапів, в залежності від жанру та інших критеріїв, але є основні, які присутні у будь-якому разі:

– виникнення ідеї;
– пошук зручних інструментів для реалізації;
– створення самої гри.

Gamedev сьогодні є дуже популярною сферою. Все більше людей сьогодні грають у відеоігри, їх зазвичай називають «геймерами» або «фанатами комп’ютерних ігор».

Є два основні типи прибутку у відеоіграх – реклама і внутрішні покупки. Рекламу зазвичай розміщують у мобільних іграх. Вона приносить пасивний прибуток. ЇЇ можна поділити на два типи:

1. Реклама, яку можна пропустити. Така реклама з'являється без бажання гравця і за її перегляд гравець нічого не отримує, при цьому гравець має змогу пропустити її через 5 секунд перегляду, або більше.

2. Реклама, яку не можна пропустити. Такий тип реклами з'являється лише коли цього захоче гравець. А щоб гравець захотів її побачити, необхідно запропонувати йому нагороду. Така реклама приносить більше прибутку ніж попередня, бо нагороду гравець отримує лише у разі, якщо додивиться до кінця.

Внутрішні покупки називають «мікротразнакції». Сьогодні складно знайти відеогру, яка б не мала можливості продати щось гравцю за реальні кошти. З їх допомогою гравець може досягти певного результату швидше або отримати ексклюзивний контент, це спеціальна мотивація гравця.

Також потрібно сказати про кіберспорт. Кіберспорт – це спортивні змагання з відеоігор, сьогодні вони розвиваються з такою ж швидкістю, як і сама сфера відеоігор. Багато країн вже вважають його офіційним видом спорту. Ця індустрія є дуже прибутковою, по аналогією з Gamedev`ом. Кожен матч підтримується спонсорами, тобто всесвітньовідомими компаніями, такими як "RedBull", а кіберспортсмени отримують заробітну плату на рівні професіональних спортсменів, що мотивує їх продовжувати кар'єру. Серед найвідоміших ігор, задіяних в кіберспорті виділяють наступні:

– League of Legends;
– Hearthstone;
– Counter Strike;
– DOTA 2.

Як і у будь-якій сфері, у Gamedev`і є певні професії:

– художники. Це творчі люди. Вони створюють візуальну частину гри, яка є дуже важливою, бо перші емоції гравця залежать саме від того, що він бачить. Крім створення графічних об'єктів до їх обов'язків входить створення

анімації для персонажів чи інших ігрових об'єктів, які будуть використовуватися у грі та малювання UI (User Interface), що перекладається з англійскьої як «Інтерфейс користувача»;

– ігрові композитори. Люди не менш творчі, ніж художники. Одною з вимог є оригінальність. Такі люди створюють унікальні саундтреки для відеоігор а також роблять озвучку, надаючи гравцям можливість почути ігрових персонажів. Без креативної музики і якісного озвучування, як і без гарної графіки, сучасна гра навряд чи буде успішною;

– програмісти. Саме вони є головною частиною будь-якого проекту. Їх задача полягає у написанні коду, який змусить гру працювати так, як потрібно. Їхніми зусиллями реалізується ігрова фізика, штучний інтелект, з яким належить битися гравцеві при грі «проти комп'ютера» та багато іншого, без них гра була б просто набором картинок та моделей;

– гейм-дизайнери. До обов'язків людини з такою професією входить грамотна побудова рівнів за допомогою графіки, яку намалювали художники, музики яку створили музиканти та коду, який написали програмісти. Для цього кожен гейм-дизайнер повинен хоча б трішки розбиратися у роботі кожної з цих професій, інакше він не зможе порозумітися з іншими членами команди;

– тестери. Коли художники намалювали усі текстури, музиканти написали саундтрек та створили озвучку, програмісти написали код, а геймдезайнери склали це в купу і видали фінальний результат, у справу вступають тестери. Їх задача полягає у виявленні недоліків гри, так званих "багів", які повинні бути виправлені до офіційного виходу гри. Після їх роботи зазвичай програмісти та гейм-дизайнери працюють над виправленням помилок у роботі програми.

Може скластися думка, що для створення гри обов'язково необхідна команда. Але мати команду для створення гри сьогодні - зовсім необов'язково, дуже багато ентузіастів сьогодні можуть виконувати функції усіх перелічених професій самотужки, але це потребує набагато більше часу, ніж робота у команді. Саме тому такі люди не беруться за величезні проекти з відкритим

ігровим світом, а створюють не менш цікаві, але простіші у реалізації ігри. Якщо виникає проблема нестачі часу, в інтернеті сьогодні є можливість замовити роботу художника чи музиканта, а роботу тестерів перекласти на своїх друзів, тобто сама людина повинна буде лише написати код та зробити з усього матеріалу гру.

1.1.2. Класифікація відеоігор

Для більш зручного орієнтування серед усього різноманіття відеоігор їх класифікують за різними ознаками.

Ось ключові параметри класифікації відеоігор:

– за кількістю гравців. Можна грати самим, з друзями чи рідними у себе вдома, або взагалі з невідомими людьми через мережу інтернет. Останній вид сьогодні є найпопулярнішим, а через пандемію коронавірусу сьогодні навіть деякі кіберспортивні змагання проводяться дистанційно;

– за ігровими платформами. Сьогодні є три основні платформи для розповсюдження відеоігор: консолі (Xbox, Play Station, Nintendo Switch), персональні комп'ютери та мобільні пристрої (Android, IOS);

– за жанром. Це ключовий фактор для ігрових студій, бо від цього залежить кількість необхідного на розробку часу, вікова категорія гри, необхідні навички та багато іншого;

– за умовами розповсюдження. Деякі ігри мають обмеження щодо регіонів їх розповсюдження через заборони зі сторони країни чи зі сторони самого розробника.

За жанром можна класифікувати відеоігри наступним чином:

Стратегії – ігри розраховані на планування стратегії для досягнення певної мети. Гравець керує військом, будує місто або розробляє план по захопленню фортеці, тобто планує свої дії і їх майбутній результат. Для досягнення результатів потребує багато часу. Приклади ігор: Герої меча та магії, Військове ремесло від Blizzard Entertainment.

Карточні ігри. Користуються популярністю серед гравців вже згаданих стратегій. Їх різноманіття також велике, і взагалі цей жанр бере свій початок давно. Приклади ігор: HearthStone, Гвинт, Uno

Раннери – ігри, які не забирають багато часу і не мають кінця, зазвичай розраховані на мобільні платформи. Під час гри персонаж біжить\летить\пливе по безкінечному ігровому світу. Для мотивації такі ігри мають рекорд. Наша гра відноситься до цього жанру. Приклади ігор: Subway surfers, Temple Run 2, Minion Rush, Crossy Road.

Симулятор – ігри, які допомагають гравцю відчути себе в різних ситуаціях, наприклад водієм трактора чи автобуса. Існує чимало під-жанрів, технічні, аркадні, спортивні, економічні, побачень та інші. Симуляторами можуть також називатись деякі представники стратегій. Приклади ігор: Clash of Clans, Pocket Troops, Boom Beach, Fallout: Shelter, серія ігор Anno.

Пригоди – ігри, в яких гравець може насолоджуватись ігровим світом. Зазвичай в таких іграх найголовнішими є атмосфера і сюжет. Персонаж подорожує, виконує завдання інших персонажів і взаємодіє з навколишнім світом. Такі ігри не розраховані на те, щоб поставити виклик гравцю, а навпаки допомагають йому відпочити. Приклади ігор: Disney's Aladdin in Nasira's Revenge, Індіана Джонс, Limbo, Mario.

Екшн – в таких іграх треба використовувати реакцію і рефлекси. Як правило екшн ігри пов’язані із битвами. Також екшн є сукупністю піджанрів, таких як: стрілялки (від 1ї або 3ї особи), драки, тактичні, стелс. Саме цей жанр є найпопулярнішим сьогодні. Приклади ігор: Wolfenstein 3D, Макс Пейн, Alien Shooter, Counter-Strike, Hitman та інші.

1.1.3. Аналіз існуючих аналогів

Для розробки власної гри було проведено аналіз популярної гри «Angry Birds 2» (рис. 1.1). Ця гра належить популярній у всьому світі серії ігор, про яку, мабуть, чув кожен.

Рис. 1.1. Гра «Angry Birds 2»

В даній грі дуже цікава основна механіка цієї гри – гравцю необхідно налаштовуючи постріл влучати в конкретну ціль, але є і деякі обмеження цієї гри.

Після проведення аналізу популярної гри «Angry Birds 2» було виявлено наступні недоліки: не завжди можна грати коли того хоче гравець, присутня реклама та мікротранзакції, ігрових рівнів кінцева кількість, неадаптованість гри під геймпад та вага у 371 МБ.

Було вирішено створити власну гру, яка б мала наступні виправлення згаданих вище недоліків: гравець може грати, коли захоче, відсутня реклама та мікротранзакції, відсутній екран завантаження, рейтинг гравців та досягнення

реалізовано за допомогою плагіну «Google Play Services», жанр гри – раннер, тобто гра є безкінечною, адаптовано під геймпад та має меншу вагу.

На рис. 1.2 можна побачити виявлені недоліки та запропоновані виправлення, які буде втілено у розробленій в рамках кваліфікаційної роботи грі. Дані останнього критерію для розробленої гри були отримані вже після створення самої гри.

Критерій	Angry Birds 2	BallBall
Чи можна грати коли захочеш?	Ні, можна грати лише якщо ти маєш життя, інакше необхідно чекати	Можна грати коли хочеш
Присутні мікротранзакції?	Присутні	Відсутні
Чи є у грі реклама?	Так, реклама присутня	Відсутня
Чи потрібно чекати перед початком гри?	Так, присутній екран завантаження	Ні, починаєш грати миттєво
Як реалізовано рейтинг гравців?	За допомогою Facebook	За допомогою Google Play Services
Як реалізовано досягнення?	Реалізовані у самій грі	За допомогою Google Play Services
Чи є у гри кінець?	Так, бо ігрових рівнів кінцева кількість	Можна грати безкінечно
Чи оптимізована гра під геймпад?	Ні	Так
Скільки важить гра?	**371 МБ**	**60 МБ**

Рис. 1.2. Переваги гри «BallBall» над грою «Angry Birds 2»

Головними з представлених критеріїв є:

– відсутність у розробленій грі реклами та внутрішніх покупок;

– вага розробленої гри майже у 6 разів менша за вагу «Angry Birds 2» (60 та 371 МБ відповідно);

– необхідність чекати і відсутність можливості грати у «Angry Birds 2» скільки того бажає гравець.

1.2. Призначення розробки та галузь застосування

Після проведення аналізу популярної гри «Angry Birds 2» було виявлено наступні недоліки: не завжди можна грати коли того хоче гравець, присутня реклама та мікротранзакції, ігрових рівнів кінцева кількість, неадаптованість гри під геймпад та вага у 371 МБ.

Було вирішено створити власну гру, яка б мала наступні виправлення згаданих вище недоліків: гравець може грати, коли захоче, відсутня реклама та мікротранзакції, відсутній екран завантаження, рейтинг гравців та досягнення реалізовано за допомогою плагіну «Google Play Services», жанр гри – раннер, тобто гра є безкінечною, адаптовано під геймпад та має меншу вагу.

Навіщо люди грають у відеоігри? Витратити час – так. Відпочити – можливо. Але більшість грає у ігри, не тільки комп'ютерні, а й реальні, щоб показати, що вони краще інших і позмагатися з іншими.

Реалізація сучасними засобами цікавої комп'ютерної гри, що призначена для розважальних цілей, задовольнить потреби користувачів – геймерів та надасть можливість отримати задоволення від процесу гри.

1.3. Підстава для розробки

Підставою для розробки кваліфікаційної роботи бакалавра на тему «Розробка комп'ютерної гри жанру раннер з використанням двигуна Unity 3D та мови програмування C#» є наказ по Національному технічному університету «Дніпровська політехніка» від __.__. 2021р. № ____-_.

1.4. Постановка завдання

Метою кваліфікаційної роботи є аналіз існуючих засобів розробки мультимедійних ігор та створення власної гри жанру раннер.

Після проведення аналізу популярної гри «Angry Birds 2» було виявлено наступні недоліки: не завжди можна грати коли того хоче гравець, присутня реклама та мікротранзакції, ігрових рівнів кінцева кількість, неадаптованість гри під геймпад та вага у 371 МБ.

Вирішено створити власну гру, яка б мала наступні виправлення згаданих вище недоліків:

– гравець може грати, коли захоче,

– відсутня реклама та мікротранзакції,

– відсутній екран завантаження,

– рейтинг гравців та досягнення реалізовано за допомогою плагіну «Google Play Services».

Жанр гри – раннер, тобто гра є безкінечною, адаптовано під геймпад та має меншу вагу.

Вихідні дані до роботи:

1. Порівняння сучасних популярних інструментальних засобів: XNA, MonoGAMe, Phaser, Construct 2, pixi.js, Blender Game Engine, Easel.JS, Turbulenz, melonJS, PandaJS, Game Maker, Unity, Unreal Engine.
2. Визначення оптимальних засобів розробки комп’ютерних ігор.
3. Написання прототипу гри.

Розроблена гра повинна бути протестована та мати кінцевий вигляд, реалізований як повноцінний додаток.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

За правилами гри гравець має влучати у вагонетку, налаштовуючи силу пострілу снаряду.

Меню гри повинно складатися з отримання допомоги,тобто довідки або пояснень правил гри, та функцій налаштування гри (звук, мова). Меню повинно надавати можливість заходити у два режими гри, а також передивлятися

рейтинг гравців.

Програма повинна реалізовувати основну механіку гри:

1. Гравець налаштовує силу пострілу.
2. Гравець стріляє за допомогою відповідної кнопки.
3. Якщо влучає, отримує певну кількість очок, якщо не влучає – втрачає одне життя. По витраченню 4х життів гравець програє, його процес зберігається, і він може почати нову гру.

1.5.2. Вимоги до інформаційної безпеки

Для надійної роботи даної комп'ютерної гри необхідно:

- використовувати ліцензійне програмне забезпечення;
- здійснювати захист від несанкціонованого доступу;
- застосовувати джерело безперебійного живлення для захисту від перепадів напруги або збоїв у живленні.

1.5.3. Вимоги до складу та параметрів технічних засобів

Виходячи з технічних характеристик і технічних умов, які передбачаються охороною праці (стомлюваність і т.д.), вимоги до складу і параметрів технічних засобів повинні бути наступні:

1. CPU не нижче 1 Ггц;
2. ОЗУ не нижче 1 Гб;
3. Монітор SVGA (підтримка режиму 1024x768, 32 bpp, 85 Hz);
4. клавіатура;
5. маніпулятор «миша».

1.5.4. Вимоги до інформаційної та програмної сумісності

Кваліфікаційна робота на тему «Розробка комп’ютерної гри жанру раннер з використанням двигуна Unity 3D та мови програмування C#» передбачає використання двигуна Unity 3D та засобів програмування C# середи розробки IDE Microsoft Visual Studio 2017, та повинна бути сумісною з операційними системами Microsoft Windows 7 та її пізнішими версіями, а також працювати під керівництвом ОС Android.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

На основі проведеного аналізу популярної гри «Angry Birds 2» та виявлених недоліків була обґрунтована необхідність створення гри «PutBall» з використанням двигуна «Unity 3D» та мови програмування «C#». Гра представляє собою ранер-симулятор, у якому необхідно влучати у вагонетку, налаштовуючи силу пострілу.

Програма реалізовує основну механіку гри:

1. Гравець налаштовує силу пострілу.
2. Гравець стріляє за допомогою відповідної кнопки.
3. Якщо влучає, отримує певну кількість очок, якщо не влучає – втрачає одне життя. По витраченню 4х життів гравець програє, його процес зберігається, і він може почати нову гру.

Меню гри складається з можливості отримання допомоги,тобто довідки та пояснень правил гри, та функцій налаштування гри (звук, мова). Меню також надає можливість заходити у два режими гри, а також передивлятися рейтинг гравців.

Жанр гри – раннер, тобто гра є безкінечною, та адаптовано під геймпад.

2.2. Опис застосованих математичних методів

2.2.1. Швидкість рівномірного руху

Найпростішим видом руху є рівномірний, при якому відношення переміщення $\vec{S}$ до його часу t, є величиною сталою, яка називається швидкістю $\vec{v}$ (рис. 2.1).

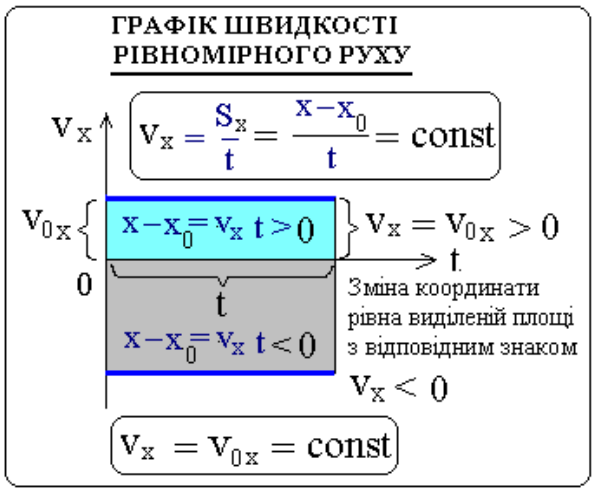

Рис. 2.1. Графік швидкості рівномірного руху

$$\vec{v} = \frac{\vec{s}}{t}. \tag{2.1}$$

Проекція вектора швидкості на обрану вісь 0x є швидкістю зміни координати по цій осі і є величиною сталою, що ілюструє відповідний графік.

$$v_x = \frac{S_x}{t} = \frac{x - x_0}{t}, \tag{2.2}$$

де v_x = const (константа)

З визначальної формули (2.2) видно, що розмірність (основна одиниця вимірювання) швидкості в Міжнародній системі одиниць (СІ) є м/с (метр розділений на секунду).

Символічно це записують так: $[v] = \frac{[S]}{[t]} = \frac{м}{с}$.

З (2.2) маємо рівняння координати рівномірного руху

$$x = x_0 + v_x t. \tag{2.3}.$$

При такому русі, залежність координати від часу є лінійною, що наочно показує графік цієї залежності (рис. 2.2).

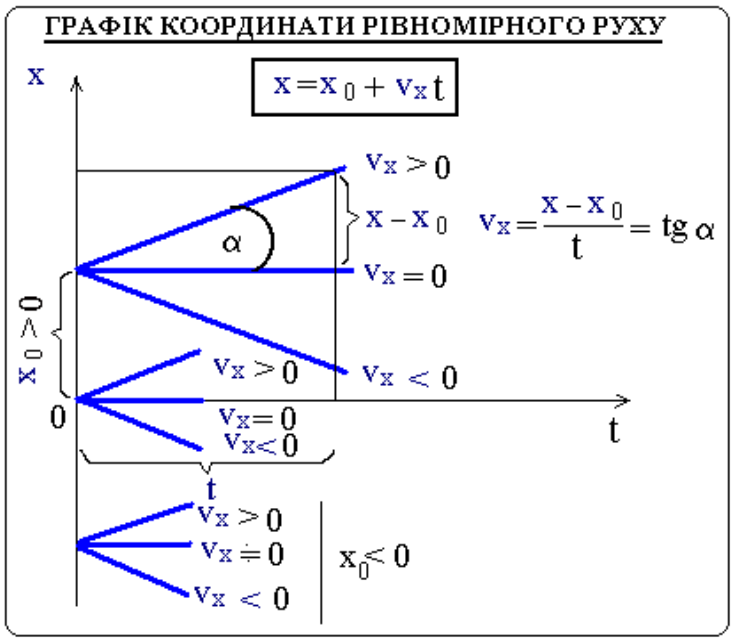

Рис. 2.2. Графік залежності координат рівномірного руху

У першому режимі гравець має влучати у вагонетку, налаштовуючи силу пострілу снаряду.

Для виконання даного етапу розробки гри досліджено рух вагонетки при трьох різних значеннях пострілу.

Обрано наступні значення сили пострілу: максимальне, середнє та мінімальне. Виходячи з формул розрахунку початкових швидкостей розроблено та заповнено таблицю даних для кожного випадку. Опір повітря при цьому випадку вважатимемо відсутнім (рис. 2.3).

Фізична величина	MAXIMUM	MEDIUM	MINIMUM
v_{x0} (м/с)	10,20	5,85	1,50
v_{y0} (м/с)	3,40	1,95	0,50
Slider.value	0,340	0,20	0,05
t_{max} (с)	3,01	2,64	2,32
x(t) (м)	$10{,}20t$	$5{,}85t$	$1{,}50t$
y(t) (м)	$6-3{,}40t-\frac{9{,}81t^2}{2}$	$6-1{,}95t-\frac{9{,}81t^2}{2}$	$6-0{,}05t-\frac{9{,}81t^2}{2}$
v_y (м/с)	$3{,}40-\frac{9{,}81t}{2}$	$1{,}95-\frac{9{,}81t}{2}$	$0{,}50-\frac{9{,}81t}{2}$
v (м/с)	10,75	6,17	1,58
$S_x(t_{max})$ (м)	30,70	15,40	3,48
y(x)	$6-\frac{x}{3}-\frac{9{,}81x^2}{104{,}04}$	$6-\frac{x}{3}-\frac{9{,}81x^2}{34{,}22}$	$6-\frac{x}{3}-\frac{9{,}81x^2}{2{,}25}$

Рис. 2.3. Таблиця даних

Дана таблиця має наступні початкові даними:

– початкові швидкості по осі oX та oY - v_{x0} та v_{y0} відповідно;

– значення сили пострілу;

– час до приземлення снаряду.

2.2.2. Моделювання пострілу

Для зручнішого користування графіками обрано три кольори – помаранчевий, зелений та блакитний для максимальної, середньої та мінімальної сили пострілу відповідно.

Першою досліджено залежність початкової швидкості від сили пострілу по вісі oX та по вісі oY (позначено їх на графіку різними кольорами) (рис. 2.4).

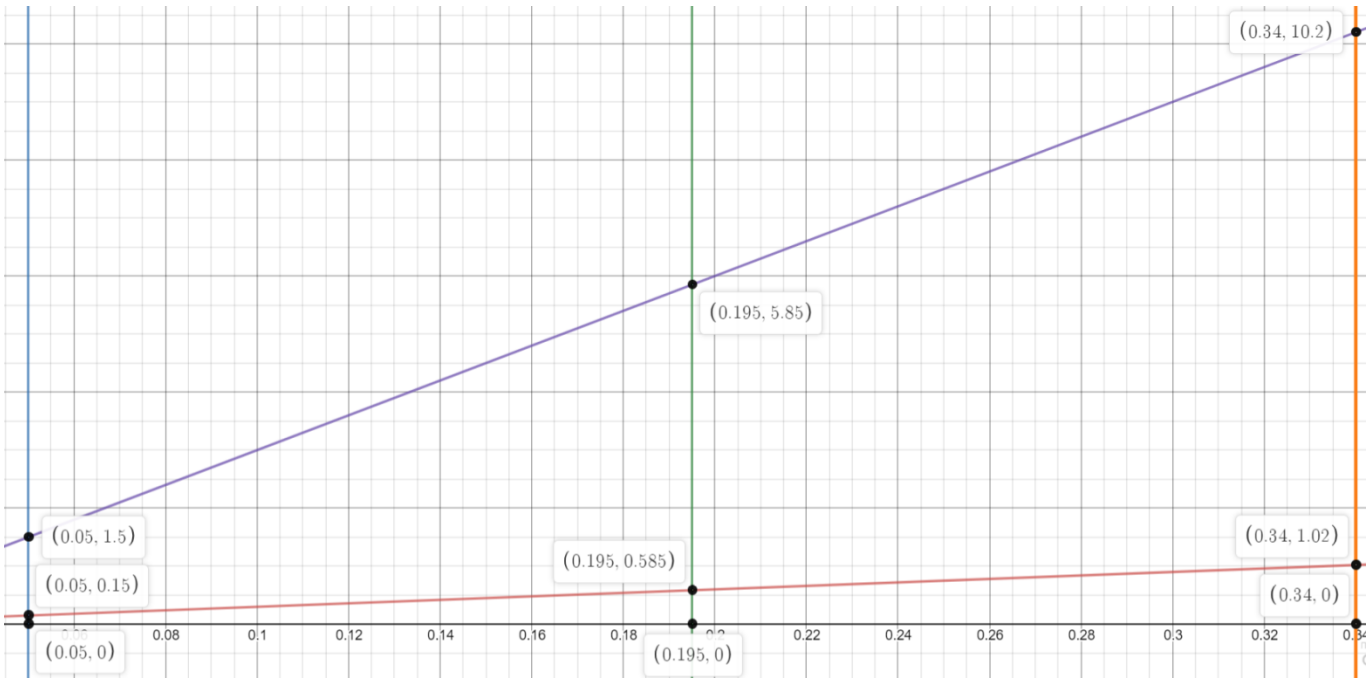

Рис. 2.4. Залежність початкової швидкості від сили пострілу по вісі oX фіолетовим кольором та по вісі oY червоним

Графік залежності швидкості по двом осям від часу, також в залежності від сили пострілу наведено на рис. 2.5 та рис. 2.6.

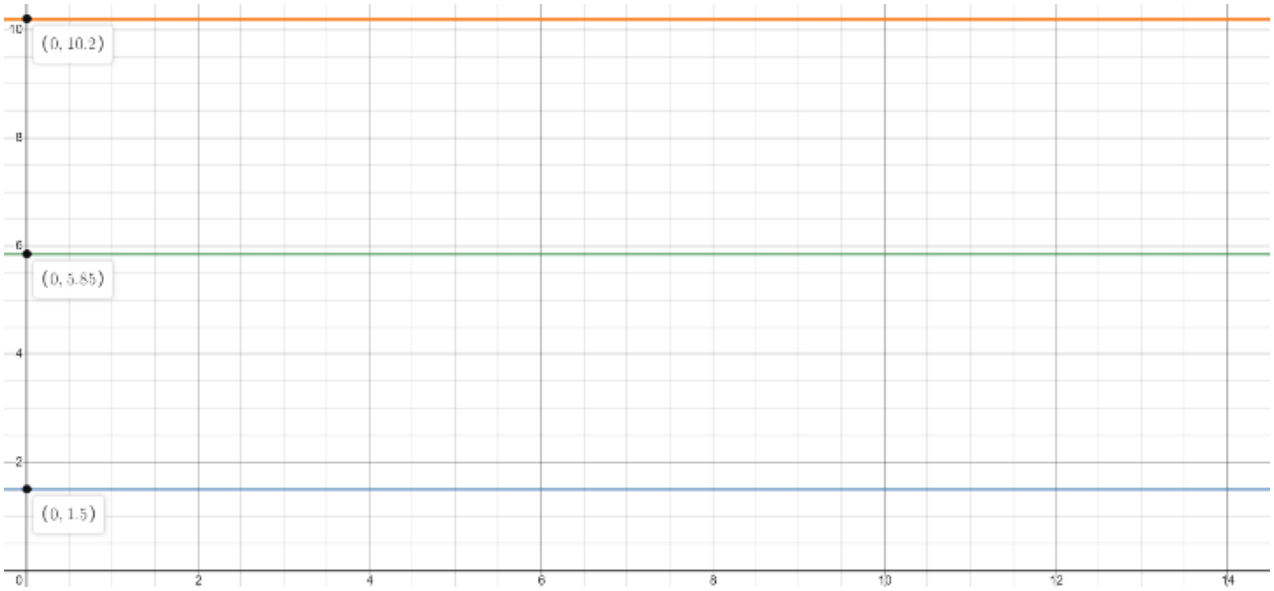

Рис. 2.5. Графік залежності швидкості від часу по осі oX

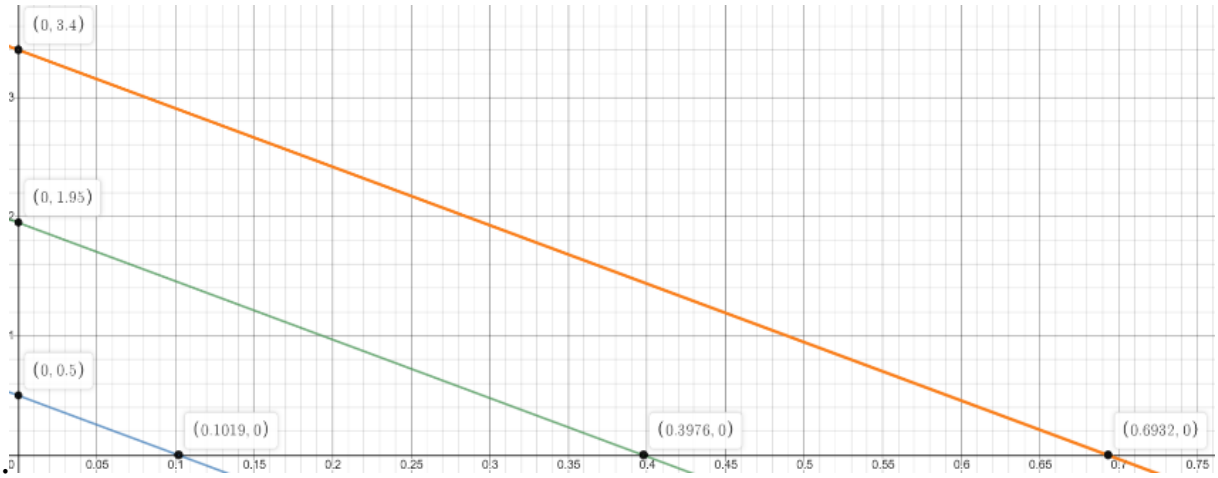

Рис. 2.6. Графіки залежності швидкості від часу по осі oY

Графік руху снаряду в залежності від сили пострілу наведено на рис. 2.7.

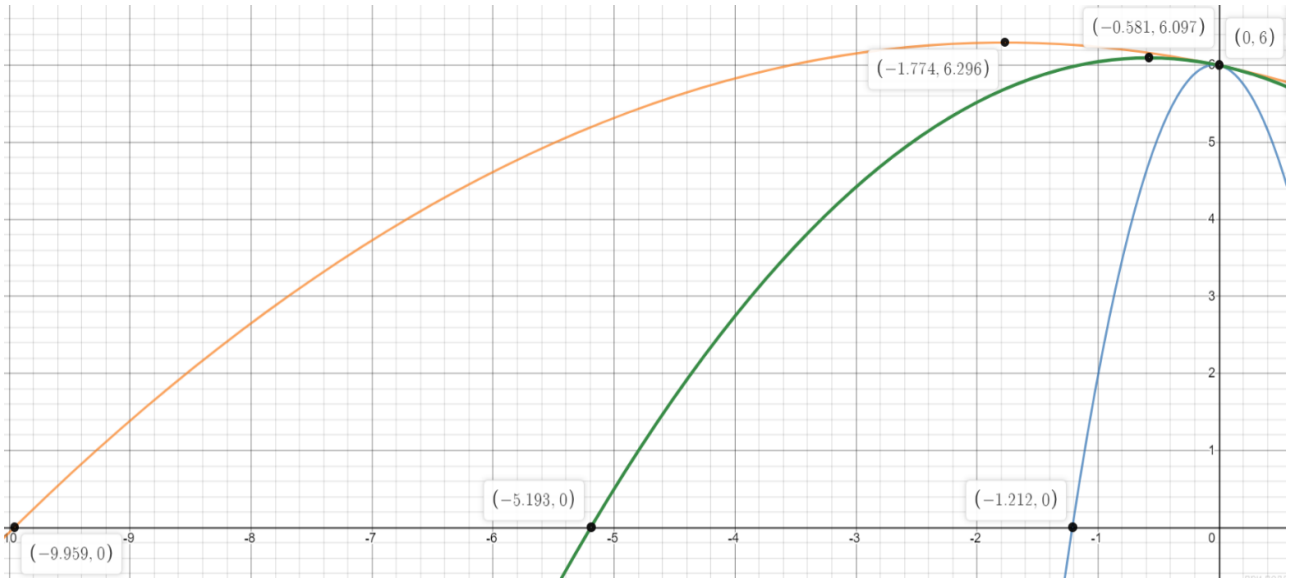

Рис. 2.7. Графіки руху снаряду, в залежності від сили пострілу

Отже, використовуючи початкові дані, використані формули залежності швидкостей по двом осям від сили пострілу та формули переміщення снаряду також в залежності від сили пострілу.

2.3. Опис використаних технологій та мов програмування

2.3.1. Двигун Unity

Unity - багатоплатформовий інструмент для розробки дво- та тривимірних застосунків та ігор, що працює на операційних системах Windows і OS X. Створені за допомогою Unity застосування працюють під системами Windows, OS X, Android, Apple iOS, Linux, а також на гральних консолях Wii, PlayStation 3 і XBox 360.Є можливість створювати інтернет застосунки за допомогою спеціального під'єднуваного модуля для браузера Unity, а також за допомогою експериментальної реалізації в межах модуля Adobe Flash Player. Застосування, створені за допомогою Unity, підтримують DirectX та OpenGL.

Unity – це кросплатформений 3D та 2D двигун, що працює на операціонних системах Windows та MacOS і дозволяє розробляти ігри майже під усі існуючі платформи та операційні системи:

- Windows;
- Android;
- MacOS;
- IOS;
- PlayStation 4;
- Nintendo Switch;
- та інші.

Сьогодні є одним з найпопулярніших рушіїв, ним користуються сотні тисяч як одиночних творців, так і ігрових студій. Через свою модульність, зручну документацію та відкритий код (у платній версії) є дуже зручним у використанні. Також підтримує DirectX та OpenGL.

В нього вбудовані такі корисні функції: внутрішній файловий провідник, вікно анімації, налаштування фізики, робота з рєстром, робота з XML файлами, підтримка написаних шейдерів, робота з рекламою та внутрішніми покупками у грі, присутнє рішення для спільної розробки – AssetStore, підтримка мережі,

генератор ландшафтів, можливість доповнювати рушій власноруч та ділитися цим з іншими.

Редактор Unity має простий Drag & Drop інтерфейс, який легко налаштовувати, що складається з різних вікон, завдяки чому можна проводити налагодження гри прямо в редакторі. Рушій підтримує три сценарних мови: C#, JavaScript (модифікація). Проект в Unity ділиться на сцени (рівні) - окремі файли, що містять свої ігрові світи зі своїм набором об'єктів, сценаріїв, і налаштувань. Сцени можуть містити в собі як, об'єкти (моделі), так і порожні ігрові об'єкти – тобто ті які не мають моделі. Об'єкти, в свою чергу містять набори компонентів, з якими і взаємодіють скрипти. У об'єктів з видимої геометрією також за замовчуванням присутній компонент Mesh Renderer, що робить їх модель їх видимою.

Також Unity підтримує фізику твердих тіл і тканини, фізику типу Ragdoll (ганчіркова лялька). У редакторі є система успадкування об'єктів; дочірні об'єкти будуть повторювати всі зміни позиції, повороту і масштабу батьківського об'єкта. Скрипти в редакторі прикріплюються до об'єктів у вигляді окремих компонентів.

При імпорті текстури в рушій можна згенерувати alpha-канал, mip-рівні, normal-map, light-map, карту відображень, проте безпосередньо на модель текстуру прикріпити не можна - буде створено матеріал, з яким буде призначений шейдер, і потім матеріал прикріпиться до моделі. Редактор Unity підтримує написання і редагування шейдерів. Крім того він містить компонент для створення анімації, анімацію також можна створити попередньо в 3D-редакторі та імпортувати разом з моделлю, а потім розбити на файли.

У Unity вбудована підтримка мережі. Ігровий рушій повністю пов'язаний із середовищем розробки. Це дозволяє випробовувати гру прямо в редакторі. Має вбудований генератор ландшафтів

2.3.2. Мова програмування C#

C# - об’єктно-орієнтована мова програмування з безпечною системою строгої типізації для платформи .NET. Розроблена та належить MicroSoft.

Її синтаксис близький до C++ та майже ідентичний з Java. Підтримує поліморфізм, первантаження операторів, вказівники на функції-члени класів, та інше. Актуальною версією є 8.0.

Її титульним компілятором є Microsoft Visual C#. Також існують інші компілятори, наприклад: Microsoft Rotor, SharpDevelop, Mono, DotGNU, DotNetAnywhere.

Спеціально для двигуна UNITY 3D під мову програмування C# було розроблено багато бібліотек. Ось основні з них, без яких не обходться жоден проект:

– UnityEngine – основна бібліотека, яка забезпечує роботу усіх компонентів у грі та яка реалізує базовий клас “MonoBehavior” від якого наслідуються усі інші класи;

– UnityEngine.SceneManagment – бібліотека, створена для роботи зі сценами в UNITY;

– UnityEngine.UI – бібліотека, створена для роботи з усім, що належить до користувацього інтерфейсу – канвас, кнопки, слайдери, важілі, картинки та багато іншого.

Також мова програмування C# дозволяє працювати з файлами, наприклад XML та реєстром, що дозволяє зберігати прогрес гравця, та усі його налаштування.

2.3.3. Інші інструменти

Для створення ігрових ассетів (компонентів гри) були використані наступні програми:

– Magica Voxel 3D – для створення 3D моделей;

– Adobe Photoshop – для створення 2D графіки та користувальницького інтерфейсу;

– FL studio – для створення саундтреку гри.

Magicavoxel - це відкритий Open Source редактор. Він містить в собі не тільки інструменти для воксельного моделювання, а й включає в себе якісний рендер движок, який дозволяє гнучко налаштувати матеріали і експортувати їх разом з запеченими текстурами і тінями.

Adobe Photoshop - графічний редактор, розроблений і поширюваний фірмою Adobe Systems. Цей продукт є лідером ринку в галузі комерційних засобів редагування растрових зображень і найвідомішим продуктом фірми Adobe. У наш час Photoshop доступний на платформах Mac OS X/Mac OS і Microsoft Windows. Ранні версії редактора були портовані під SGI IRIX, але офіційна підтримка була припинена, починаючи з третьої версії продукту. Для версії CS і CS6 можливий запуск під Linux за допомогою альтернативи Windows API - Wine.

Photoshop головним чином призначений для редагування цифрових фотографій та створення растрової графіки. Особливості Adobe Photoshop полягають у багатому інструментарії для операції створення і обробки зображень, високій якості обробки графічних зображень, зручності й простоті в експлуатації, широких можливостях до автоматизації обробки растрових зображень, які базуються на використанні сценаріїв, механізмах роботи з кольоровими профілями, які допускають їх втілення в файли зображень з метою автоматичної корекції кольорових параметрів при виводі на друк для різних пристроїв, великому наборі команд фільтрації, за допомогою яких можна створювати найрізноманітніші художні ефекти.

Базові інструменти редагування дозволяють змінювати тон, насиченість зображення, обтинати його, накладати фотофільтри, виправляти перспективу тощо. Photoshop підтримує так звані шари - прозорі області зображення, на яких розміщуються елементи фотомонтажу, текст, геометричні фігури. Програма містить інструменти для роботи з текстом і нескладними фігурами, дозволяє

малювати робочі контури, задавати текстам і фігурам стилі оформлення. Для роботи з окремими фрагментами зображення передбачені різні типи виділення: за фігурою, в режимі «малювання» зони виділення, за діапазоном кольорів тощо. Існують різноманітні фільтри для деформації та стилізації зображення, такі як фільтри розмиття, імітації різних художніх технік. Photoshop також містить інструменти для цифрового живопису, зокрема набори пензлів. Користувач може змінювати їх розмір, кут нахилу, колір. Підтримується встановлення сторонніх пензлів, стилів, шрифтів, палітр. Попри те, що спочатку програма була розроблена як редактор зображень для поліграфії, в наш час[коли?] вона широко використовується і у веб-дизайні. У більш ранній версії була включена спеціальна програма для цих цілей - Adobe ImageReady, яка була виключена з версії CS3 за рахунок інтеграції її функцій в сам Photoshop, а також включення в лінійку програмних продуктів Adobe Fireworks, що перейшло у власність Adobe після придбання компанії Macromedia.

Photoshop тісно пов'язаний з іншими програмами для обробки медіафайлів, анімації та іншої творчості. Спільно з такими програмами, як Adobe ImageReady (програма скасована у версії CS3), Adobe Illustrator, Adobe Premiere, Adobe After Effects і Adobe Encore DVD, він може використовуватися для створення професійних DVD, забезпечує засоби нелінійного монтажу і створення таких спецефектів, як фони, текстури і т. д. для телебачення, кінематографу і всесвітньої павутини. Основний формат Photoshop, PSD, може бути експортований і імпортований всіма програмними продуктами, переліченими вище. Photoshop CS підтримує створення меню для DVD. Спільно з Adobe Encore DVD, Photoshop дозволяє створювати меню або кнопки DVD. Photoshop CS3 у версії Extended підтримує також роботу з тривимірними шарами.

Підтримується обробка зображень, як з традиційною глибиною кольору (8 біт, 256 градацій яскравості на канал), так і з підвищеною (16 біт, 65536 відтінків в кожному каналі). Можливе збереження у файлі додаткових елементів, як то: напрямних (Guide), каналів (наприклад, каналу прозорості —

Alpha channel), шляхів обтравки (Clipping path), шарів, що містять векторні і текстові об'єкти. Файл може включати колірні профілі (ICC), функції перетворення кольору (transfer functions).

Photoshop підтримує такі колірні моделі або способи опису кольорів зображення (в нотації самої програми - режим зображення): RGB, LAB, CMYK, Grayscale, Bitmap, Duotone, Indexed, Multichannel. Через високу популярність Photoshop підтримка його формату файлів, PSD, була реалізована в його основних конкурентів, таких, як Macromedia Fireworks, Corel PHOTO-PAINT, Pixel image editor, WinImages, GIMP, Jasc Paintshop Pro і т. д.

FL Studio - редактор-секвенсер для написання музики, створений 1997 року програмістом Дідьє Дембреном (також відомим під псевдонімом «gol»). Музика створюється шляхом запису і зведення (звукозапису) аудіо-, або MIDI-матеріалу. Готова композиція може бути записана у файл з розширенням WAV, MP3 або OGG. Програма написана мовою програмування Delphi

У програмі міститься 4900 звукових ефектів, які допомагають як складати, так і редагувати музику.

2.4. Опис структури системи та алгоритмів її функціонування

2.4.1. Етапи розробки гри

Процес розробки для більш зручного орієнтування поділяється на такі етапи:

- розробка ідеї;
- створення графіки (як 3D моделей, так і елементів інтерфейсу);
- реалізація основної механіки гри (з використанням мови C#);
- випуск гри у Google Play;
- доповнення гри інтерактивним навчанням;
- локалізація гри;
- додання у гру Google Play Services;
- тестування серед геймерів та опрацювання їх заперечень і побажань.

2.4.1.1. Розробка ідеї

Головною метою етапу розробки ідеї було створення нового продукту, не схожого на інші, саме тому вирішено обрати як жанр раннер-симулятор. Симулюватись в даній програмі будуть постріли, а так як гра не просто симулятор, а ще й раннер будо вирішено пустити вагонетку по колу і запропонувати гравцеві попасти в неї. Фрагмент коду, в якому надається імпульсу снаряду, в залежності від сили пострілу наведено на рис. 2.8.

```
void Start()
{

    _firstBool = false;
    _secondBool = false;
    _thisRigidBody = GetComponent<Rigidbody>();
    _thisRigidBody.velocity = new Vector3(0, 0, 0);
    slid = GameObject.Find("_FireSlider").GetComponent<Slider>();

    _thisRigidBody.AddForce(new Vector3(slid.value * sila * 3, slid.value * sila, 0f), ForceMode.Impulse);

}
```

Рис. 2.8. Фрагмент коду, у якому надається імпульс снаряду, в залежності від сили пострілу

2.4.1.2. Створення графіки гри

Для створення красивої графіки було обрано програму під назвою «Magica Voxel». Як зрозуміло з назви, вона дозволяє створювати воксельні (тобто піксельні, але у трьох вимірах) моделі, легко їх експортувати та використовувати вже у UNITY. У цій програмі були створені всі 3D моделі, присутні у грі: башту, карту місцевості, дерева, ядро, вагонетку, кристал, гори та інше (рис. 2.9).

Рис. 2.9. Зображення вексельних моделей

Для створення елементів інтерфейсу було використано Adobe Photoshop. Створивши кнопки, важілі, іконки та слайдери їх також експортували у UNITY (рис. 2.10).

Рис. 2.10. Елементи інтерфейсу

2.4.1.3. Реалізація основної механіки гри

Програма реалізовує основну механіку гри:

1. Гравець налаштовує силу пострілу.
2. Гравець стріляє за допомогою відповідної кнопки.
3. Якщо влучає, отримує певну кількість очок, якщо не влучає – втрачає одне життя. По витраченню 4х життів гравець програє, його процес зберігається, і він може почати нову гру.

Меню гри складається з можливості отримання допомоги,тобто довідки та пояснень правил гри, та функцій налаштування гри (звук, мова). Меню також надає можливість заходити у два режими гри, а також передивлятися рейтинг

гравців.

Жанр гри – раннер, тобто гра є безкінечною.

Для відтворення цих пунктів було використано бібліотеки UnityEngine, UnityEngine.UI та UnityEngine.SceneManagment (рис. 2.11).

Рис. 2.11. Червоним кольором – налаштування сили пострілу, синім кольором – кнопка пострілу, жовтим кольором – життя гравця, рожевим – набрані гравцем очки

Також було розроблено спеціальний екран програшу (рис. 2.12), де гравець може побачити свій рекорд, набрану кількість очок, вийти у меню чи почати нову гру.

Рис. 2.12. Зеленим кольором – кнопка «почати нову гру», синім кольором– кнопка «головне меню», червоним кольором – кількість набраних очок, жовтим кольором – рекорд

2.4.2. Реалізація прототипу гри на Unity

Unity studio – це окрема середа розробки з набором вбудованих редакторів та інструментів для відладки гри. Є можливість запустити гру а самій студії, слідкувати за змінними , емулювати дії гравця, тощо (рис. 2.12).

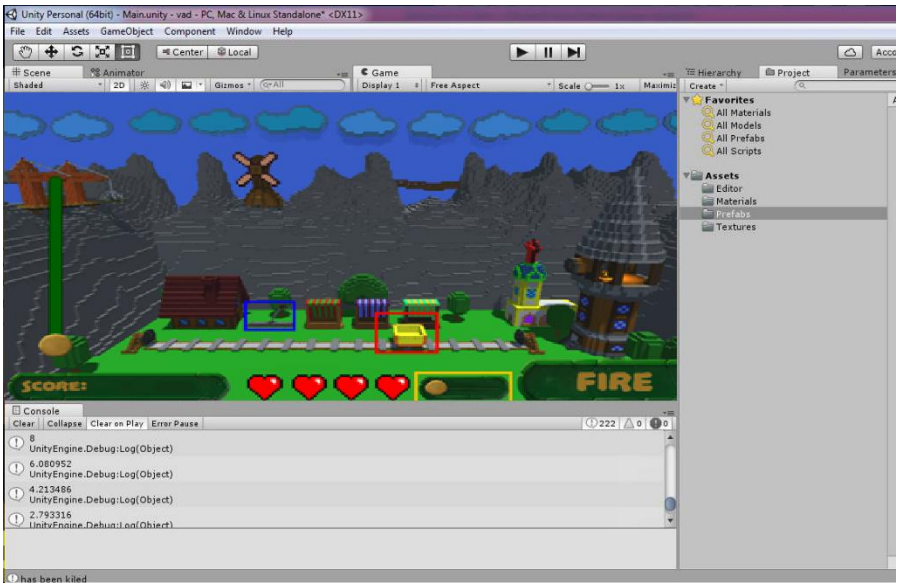

Рис.2.12. Головний екран Unity studio, редактор сцени та предпоказ ігрового екрану

Щоб додати графічні ресурси варто лише приєднати їх до проекту у спеціальному менеджері проекту. Студія автоматично розцінює графічні файди як текстури, проте якщо це набір спрайтів потрібно лише змінити властивість об‘єкту на спрайт. Також присутній редактор спрайтів, де можна нарізати спрайти автоматично, чи вручну, задати центр для кожного спрайту, тощо (2.13).

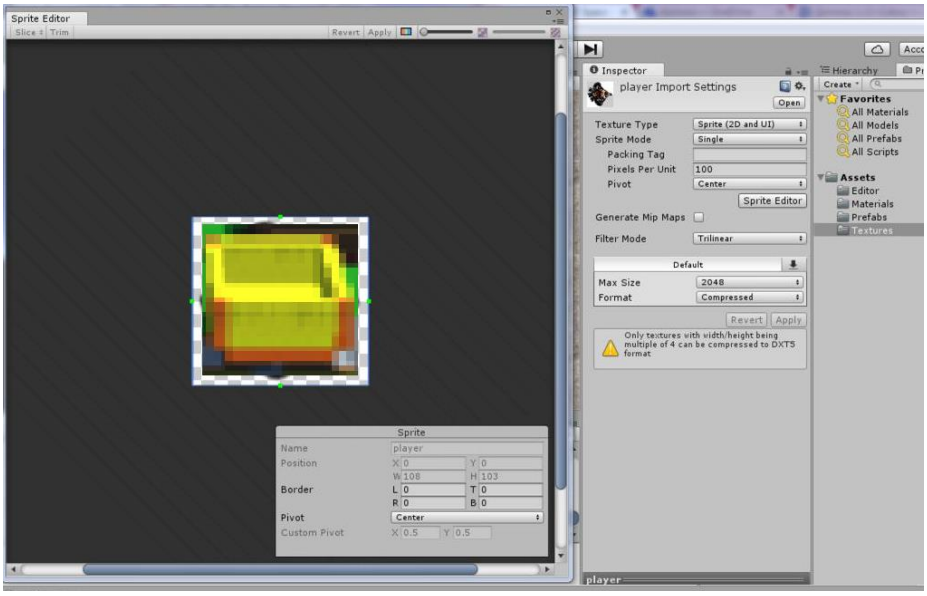

Рис. 2.13.Робота з графічними ресурсами в Unity

В Unity э декілька вбудованих рушіїв для 2D та 3D та декілька способів взаємодії з ними. Перш за все потрібно додати компонент Collider до об‘єкту гри, тип Collider буде визначати тип рушія, для прототипу використаємо

Physics 2D Collider. Цей колайдер буде використовуватись для визначення претину та доторкання об‘єктів з іншими колайдерами, додаткові налаштування triggers дозволяє обмету проходити через інші об‘єкти проте так само викликати подію накладіння, напроти Cinematic objects буде зупиняти об‘єкти що доторкаються до нього , проте не будуть викликати подію накладання якщо будуть рухатись самі (рис. 2.13).

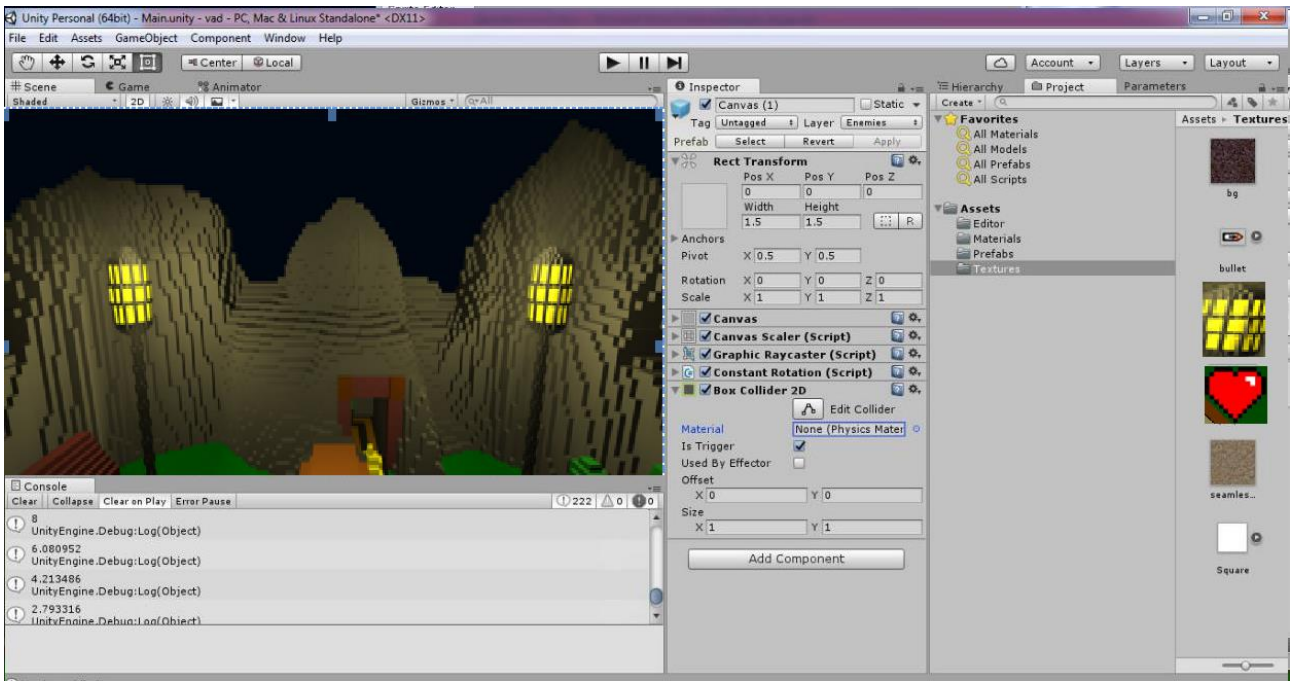

Рис. 2.13. Налаштування Collider в Unity

Ще одною важливою деталлю налаштування колізії є рівні (Layers) можна назначити певним об‘єктам певні рівні. Об’єкти між різними рівнями не взаємодіють, до того ж можна налаштувати колізію між об‘єктами на одному рівні.

Робота з логікою гри проходить за допомогою написання скриптів та додаванню цих скриптів до сцени чи обраного об‘єкту. Головна концепція скриптів це використання певних явищ чи методів з періодичним викликом. Таким чином цей скрипт можна приєднати до декількох об‘єктів та декілька скриптів до одного. Скажімо скрипт ,що під час кожного виклику методу Update перевіряє чи не знизився показник здоров‘я до 0, і якщо так, то знищує його.

Скрипти пишуться на мові C# в MS Visual Studio з якою інтегрується проект, створений в Unity studio. Таким чином стає доступним весь арсенал відладки Visual Studio при роботі з грою (рис. 2.14).

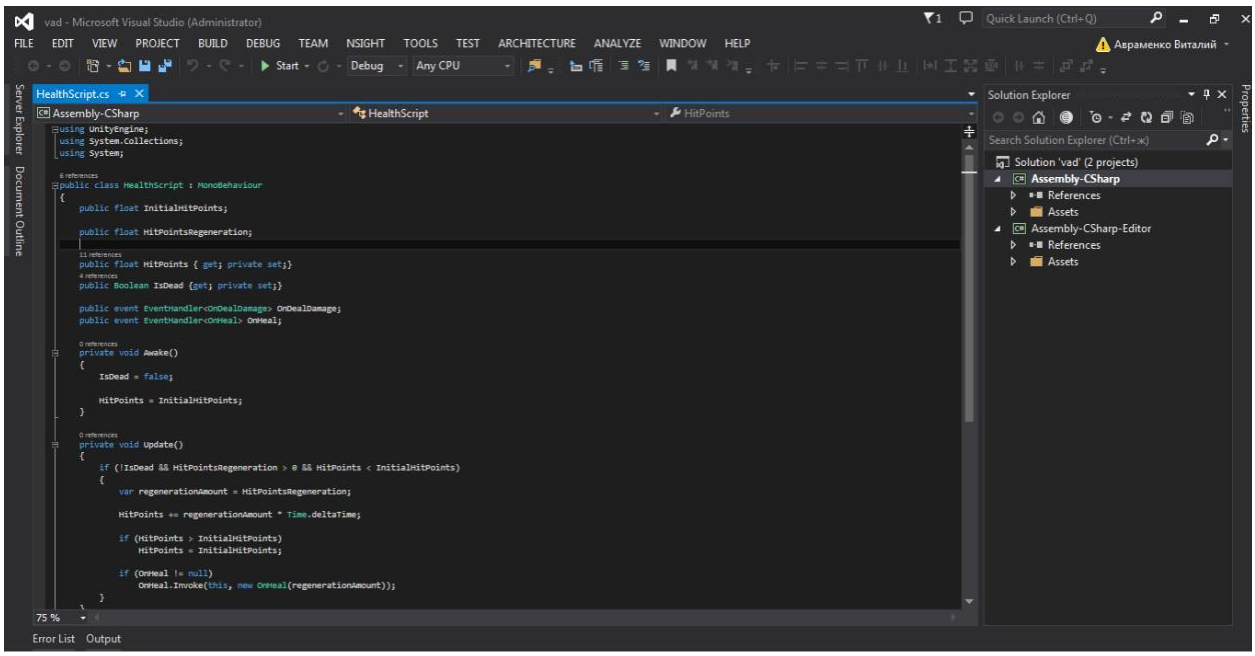

Рис. 2.14. Скрипт для Unity в Visual Studio

В Unity існує два методи, призначення яких обновлювати данні, Fixedupdate та Update. Різниця в тому, що Fixedupdate викликається зі сталою періодичністю в приблизно 2 мілісекунди і використовується для обробки фізики, зміни швидкості, тощо. Тоді як Update викликається кожного разу перед оновленням фрейму, іноді період цього виклику більше 2 мілісекунд іноді менше. Для обробки сингалів від гравця використовується саме методі Update, адже він викличеться одразу ж після отримання сигналу. В цьому методі ми маємо доступ до об‘єкту input, з його допомогою дізнаємося про стан клавіатури та очікуємо натискання якоїсь певної клавіші. Або натомість використати більш гнучке рішення, звернувшись до властивості input під назвою Axes – своєрідний рівень абстракції до якого можна прив‘язати логіку гри. Таким чином незалежно від того, буде використовуватися клавіатура, джойстик чи навіть тачскрін як пристрій вводу, гра буде коректно працювати. Настроїти об‘єкти Axes можна в спеціальному менеджері, задати ім‘я, та сигнали за замовчуванням (рис. 2.15).

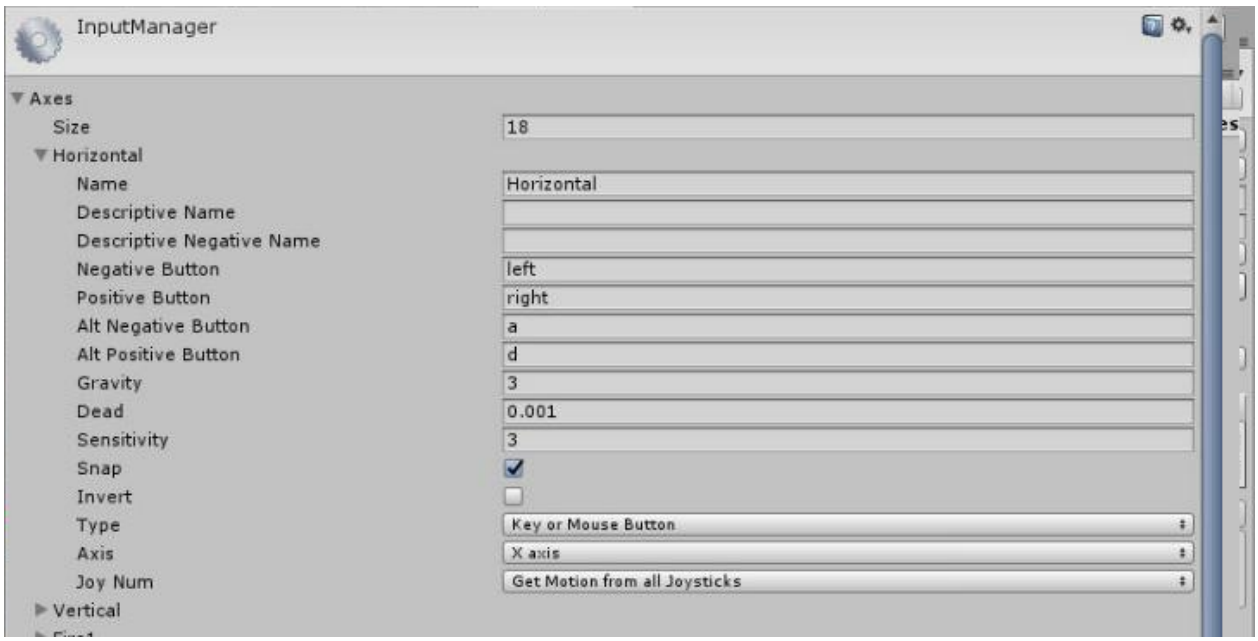

Рис. 2.15. Менеджер налаштування об‘єктів Axes в Unity

Цікавість першого режиму додатково полягає в двох важілях та декількох станах вагонетки.

Перший важіль знаходиться у користувацькому інтерфейсі (рис. 2.16). За його допомогою можна змінювати стиль гри.

Рис. 2.16. Синім кольором – важіль у ігровому світі, красним – золота вагонетка, жовтим – важіль у користувацькому інтерфейсі

Зазвичай вагонетка рухається вперед, чи назад. Якщо гравець попаде в неї, він отримає 2 очки. У другому стилі вагонетка стоїть на місці, при кожному попаданні змінює свою позицію та дає гравцю одне очко.

З часом вагонетка різко змінюватиме свою швидкість на таку, при якій практично неможливо попасти. Щоб це виправити знадобиться другий важіль, який розташований у ігровому світі. Це додає контрасту до гри та не дає гравцю засумувати.

Зазвичай вагонетка при попаданні дає 2 чи 1 очко, але через деякий час після початку гри вона може стати золотою на 4 секунди. Якщо гравець встигне влучити в неї, то отримає в 5 разів більше очок, 10 чи 5 відповідно. Це додає азарту грі та таким чином гравець може втратити життя, тобто це також ускладнює гру.

Золота і прискорена вагонетки можуть співпадати, це ще більше ускладнює гру.

Також було додано у гру другий режим (рис. 2.17).

Рис. 2.17. Другий режим гри. Червоним кольором – кнопки повороту, блакитним кольором – кнопка пострілу, рожевим кольором – життя гравця та жовтим кольором – рахунок гравця.

Його суть теж полягає у тому, щоб влучити у вагонетку, але тепер замість сили пострілу гравцю необхідно обрати позицію, а саме оберт башти, яка знаходиться всередині, щоб попасти у вагонетку, яка кружляє навколо гравця. Так як цей режим є простішим за перший, за одне влучення гравець отримує по 1 очку. Час від часу вагонетка може змінювати напрямок свого руху, тим самим підвищувати інтерес до гри.

Прогрес першого і другого режимів зберігаються окремо.

Одним з найважливіших етапів було створення інтерактивного навчання, бо якщо гравець не зрозуміє як грати – він просто видалить гру. Текстовий варіант навчання не є гарним рішенням, бо гравець хоче грати, хоче руху і тому зосереджуватись на тексті не буде. Саме тому було використано мінімум тексту, а більше практики. У процесі навчання гравець побачить усі ігрові механіки та навчиться ними користуватися (рис. 2.18).

Рис. 2.18. Інтерактивне навчання

Ще одним з важливих пунктів була локалізація, тобто переклад гри на інші мови. Якщо гравець не зрозуміє гру, він не витрачатиме час на її опанування, а просто її видалить. Також на місці гравця куди зручніше якщо не ти сам обираєш мову, а за тебе це робить сама гра. За допомогою можливостей

мови програмування C# кожного разу отримуємо мову, яку використовує операційна система гравця, і в залежності від неї обирається переклад. Це робиться при кожному вході у гру, тобто якщо гравець змінить мову операційної системи, то і мова гри зміниться.

Плагін від Google, який має назву “Play Services”, дозволяє взаємодіяти гравцям у іграх. Він прив’язується до Google-аккаунту гравця і зберігає усе, що необхідно. Сьогодні дуже багато мобільних ігор використовують його.

У нашій грі було використано цей плагін для додання загальної таблиці лідерів – рейтингу гравців, який може подивитися кожен і який оновлюється після кожної завершеної гри. Щоб переглянути його, необхідно натиснути кнопку у головному меню (рис. 2.19).

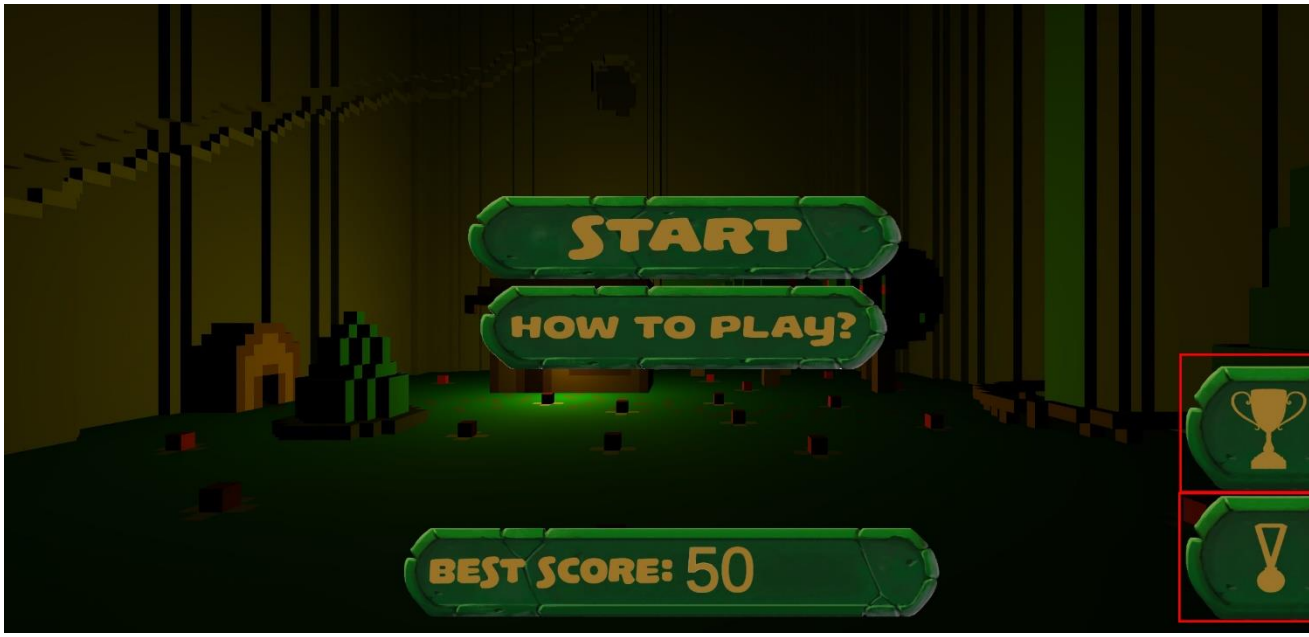

Рис. 2.19. Кнопка відкриття рейтингу гравців та досягнень відповідно

Також Google Services знадобилися щоб стимулювати гравців іншим методом – досягнення. При отриманні досягнення гравець отримує очки досвіду у Play іграх, що підвищує його рівень. У грі є досягнення за кількість набраних очок і одне секретне завдання, для активації якого необхідно знайти у першому режимі конкретне дерево і натиснути по ньому.

Тобто використання Google Play Services підвищує інтересу до гри у гравців (рис. 2.20). Рейтинг і досягнення створені для обох режимів, стимулюють користувачів до покращення результатів.

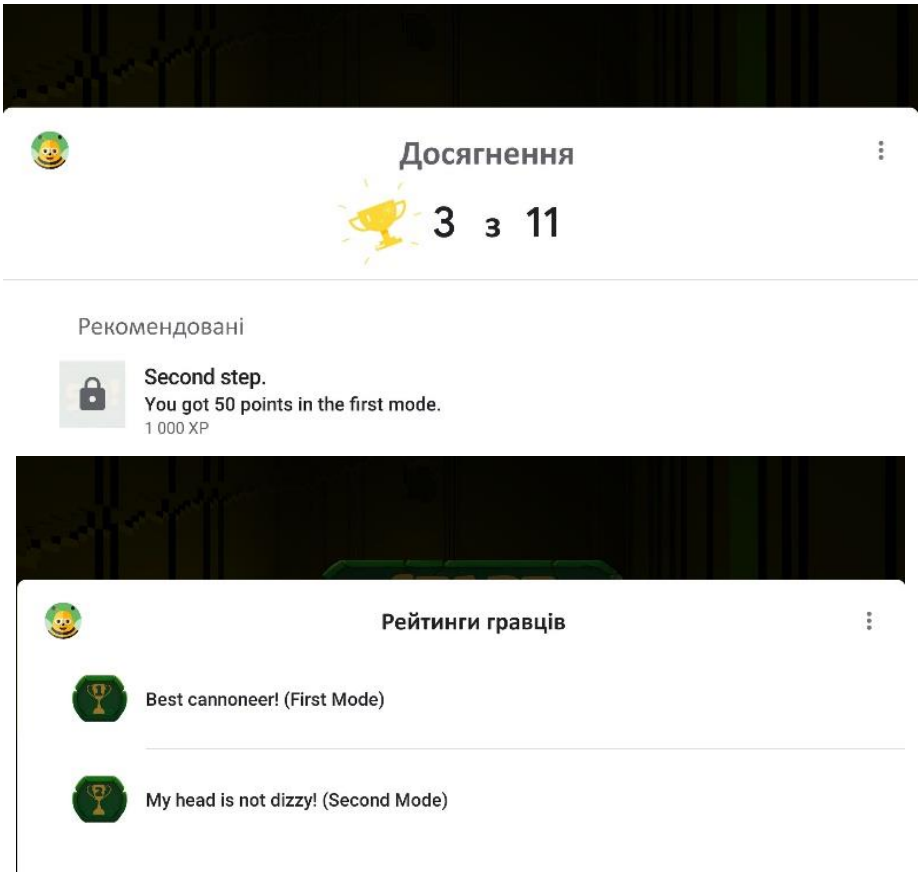

Рис. 2.20. Досягнення та рейтинги гравців

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Вхідними даними даної програми є вибір одного з режимів гри, завдання налаштувань звуку та мови інтерфейсу, завдання параметрів сили пострілу та координат траєкторії пострілу.

Вихідними даними програми є відображення ігрового процесу, прорахунок траєкторії руху снаряду та його зображення, відображення балів та життів ігрока.

2.6. Опис роботи розробленої системи

2.6.1. Використані технічні засоби

Для розробки даної гри використовувалися наступні технічні засоби:

– процесор: Mobile DualCore Intel Core i3-350M, 2266 MHz ;
– відеокарта: ATI Mobility Radeon HD 5650;
– пам’ять: 700Mb;
– ОС: Windows 7.

2.6.2. Використані програмні засоби

Для досягнення поставленої мети було обрано наступні програми: Unity 3D, Adobe Photoshop,«Magica Voxel 3D, Visual Studio 2017, додаток Google Play Developer Console, мова програмування C#.

2.6.3. Виклик та завантаження програми

Для початку гри необхідно завантажити додаток BallBall на телефон з операційною системою Android та виконати розпаковку його архіву. Після цього на екрані з’явиться іконка з назвою гри BallBall, натиснув на яка відбудеться вхід в гру.

2.6.4. Опис інтерфейсу користувача

Жанр гри – раннер, тобто гра є безкінечною.

Головна механіка гри:

1. Гравець налаштовує силу пострілу.
2. Гравець стріляє за допомогою відповідної кнопки.

3. Якщо влучає, отримує певну кількість очок, якщо не влучає – втрачає одне життя. По витраченню 4х життів гравець програє, його процес зберігається, і він може почати нову гру.

Меню гри складається з можливості отримання допомоги, тобто довідки та пояснень правил гри, та функцій налаштування гри (звук, мова). Меню також надає можливість заходити у два режими гри, а також передивлятися рейтинг Гравців (рис. 2.21).

Рис. 2.21. Головне меню гри

Одним з найважливіших етапів було створення інтерактивного навчання, бо якщо гравець не зрозуміє як грати – він просто видалить гру. Текстовий варіант навчання не є гарним рішенням, бо гравець хоче грати, хоче руху і тому зосереджуватись на тексті не буде. Саме тому було використано мінімум тексту, а більше практики. У процесі навчання гравець побачить усі ігрові механіки та навчиться ними користуватися (рис. 2.22).

Рис. 2.22. Допомога гравцю

Також було розроблено спеціальний екран програшу, де гравець може побачити свій рекорд, набрану кількість очок, вийти у меню чи почати нову гру (рис. 2.23).

Рис. 2.23. Меню гри

Цікавість першого режиму полягає в двох важілях та декількох станах вагонетки (рис. 2.24).

Рис. 2.24. Процес гри першого режиму

Перший важіль знаходиться у користувацькому інтерфейсі. За його допомогою можна змінювати стиль гри.

Зазвичай вагонетка рухається вперед, чи назад. Якщо гравець попаде в неї, він отримає 2 очки. У другому стилі вагонетка стоїть на місці, при кожному попаданні змінює свою позицію та дає гравцю одне очко.

З часом вагонетка різко змінюватиме свою швидкість на таку, при якій практично неможливо попасти. Щоб це виправити знадобиться другий важіль, який розташований у ігровому світі. Це додає контрасту до гри та не дає гравцю засумувати.

Зазвичай вагонетка при попаданні дає 2 чи 1 очко, але через деякий час після початку гри вона може стати золотою на 4 секунди. Якщо гравець встигне влучити в неї, то отримає в 5 разів більше очок, 10 чи 5 відповідно. Це додає азарту грі та таким чином гравець може втратити життя, тобто це також ускладнює гру.

Золота і прискорена вагонетки можуть співпадати, це ще більше ускладнює гру (рис. 2.25).

Рис. 2.25 Процес гри першого режиму

Також було додано у гру другий режим (рис. 2.26).

Рис. 2.26. Процес гри другого режиму

Суть гри другого режиму теж полягає у тому, щоб влучити у вагонетку, але тепер замість сили пострілу гравцю необхідно обрати позицію, а саме оберт башти, яка знаходиться всередині, щоб попасти у вагонетку, яка кружляє навколо гравця. Так як цей режим є простішим за перший, за одне влучення гравець отримує по 1 очку. Час від часу вагонетка може змінювати напрямок свого руху, тим самим підвищувати інтерес до гри (рис. 2.27).

Рис. 2.27. Закінчення гри

Прогрес першого і другого режимів зберігаються окремо.

При отриманні досягнення гравець отримує очки досвіду у Play іграх, що підвищує його рівень. У грі є досягнення за кількість набраних очок і одне секретне завдання, для активації якого необхідно знайти у першому режимі конкретне дерево і натиснути по ньому (рис. 2.28).

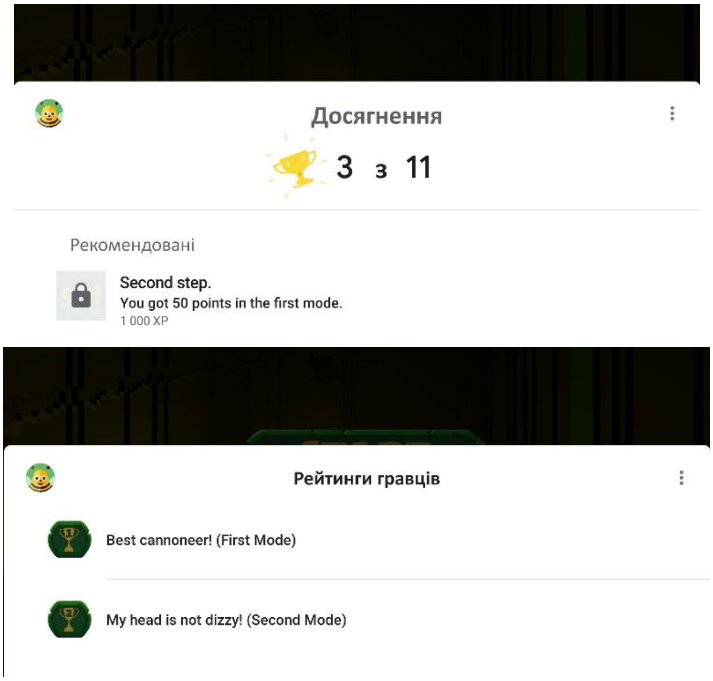

Рис. 2.28. Досягнення та рейтинги гравців

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Вихідні дані розробки програмного забезпечення:

- передбачуване число операторів – 860;
- коефіцієнт складності програми – 1,2;
- коефіцієнт кореляції програми в ході її розробки - 0,1;
- середня годинна заробітна плата програміста, грн/год – 100;
- вартість машино-години ЕОМ, грн/год – 5.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_о + t_и + t_а + t_п + t_{отл} + t_д \text{, людино-годин,} \quad (3.1)$$

де $t_о$ – витрати праці на підготовку й опис поставленої задачі (приймається 50);

$t_и$ – витрати праці на дослідження алгоритму рішення задачі;

$t_а$ – витрати праці на розробку блок-схеми алгоритму;

$t_п$ – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C(1 + p), \tag{3.2}$$

де q – передбачуване число операторів;

С– коефіцієнт складності програми;

р – коефіцієнт кореляції програми в ході її розробки.

$$Q = 860 \cdot 1{,}2 \cdot (1 + 0{,}1) = 1135;$$

Витрати праці на вивчення опису задачі $t_и$ визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75...85)K}, \text{ людино-годин}, \tag{3.3}$$

де В – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі, В=1.2 ... 1.5;

К – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. до 2 – 0,8;

$$t_u = \frac{1135 \cdot 1{,}2}{85 \cdot 0{,}8} = 20, \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20...25)K}; \tag{3.4}$$

$$t_a = \frac{1135}{25 \cdot 0{,}8} = 57, \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)K}; \quad (3.5)$$

$$t_n = \frac{1135}{25 \cdot 0{,}8} = 57 \text{, людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4...5)K}; \quad (3.6)$$

$$t_{отл} = \frac{1135}{5 \cdot 0{,}8} = 284 \text{, людино-годин,}$$

- за умови комплексного налагодження завдання:

$$t_{отл}^{к} = 1{,}2 \cdot t_{отл}; \quad (3.7)$$

$$t_{отл}^{к} = 1{,}2 \cdot 284 = 340 \text{, людино-годин}$$

Витрати праці на підготовку документації:

$$t_д = t_{др} + t_{до}; \quad (3.8)$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису

$$t_{др} = \frac{Q}{(15...20)K}; \quad (3.9)$$

$$t_{др} = \frac{1135}{20 \cdot 0{,}8} = 71 \text{, людино-годин.}$$

$t_{до}$ – трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0{,}75 \cdot t_{др}; \quad (3.10)$$

$$t_{до} = 0{,}75 \cdot 71 = 53, \text{ людино-годин.}$$

$$t_{д} = 71 + 53 = 124, \text{ людино-годин.}$$

Отримаємо трудомісткість розробки програмного забезпечення:

t = 50 +20 + 57 +57 + 284 +124 = 592, людино-годин.

В результаті розраховано, що в загальній складності необхідно 592 людино-годин для розробки даного програмного забезпечення.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ Кпо включають витрати на заробітну плату виконавця програми Зз/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн,} \quad (3.11)$$

де $З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн,} \quad (3.12)$$

де t – загальна трудомісткість, людино-годин;

$С_{ПР}$ – середня годинна заробітна плата програміста, грн/година

$$Ç_{Çï} = 592 \cdot 100 = 59200, \text{грн.}$$

$З_{МВ}$ – Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \cdot C_М \text{, грн,} \qquad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год.

$C_{Мч}$ – вартість машино-години ЕОМ, грн/год.

$$З_{МВ} = 283 \cdot 5 = 1419 \text{, грн.}$$

$$\hat{E}_{ïï} = 59200 + 1419 = 60619 \text{ , грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p} \text{, міс.} \qquad (3.14)$$

де B_k - число виконавців;

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні Fр=176 годин).

$$T = \frac{592}{1 \cdot 176} = 3{,}4 \text{ міс.}$$

Таким чином, очікувана тривалість розробки складе 3,4 місяця, а витрати на створення програмного забезпечення 60619 грн.

ВИСНОВКИ

Gamedev розвивається дуже швидко. Серед людей, які належать цій сфері прийнято розробку гри ділити на етапи, в залежності від того, хто її створює, але є декілька основних. До них відносяться: виникнення ідеї, пошук зручних інструментів для реалізації та створення самої гри.

Серед ігрових рушіїв у колі незалежних розробників, останнім часом, найбільшою популярністю користується Unity. Він повністю задовольняє вирішення поставленого завдання за своїми базовими характеристиками до того ж має потужну підтримку товариства, що значно знижує поріг входження.

Метою кваліфікаційної роботи є аналіз існуючих засобів розробки мультимедійних ігор та створення власної гри жанру раннер.

Для досягнення поставленої мети було обрано наступні програми: «Unity 3D», «Adobe Photoshop», «Magica Voxel 3D», «Visual Studio», додаток «Google Play Developer Console».

Для гри було створено два ігрових режими, кожен з яких доповнено унікальними механіками. Основна механіка обох режимів – вагонетка, яка рухається, і в яку необхідно влучити.

Практичне значення роботи полягає в реалізації сучасними засобами цікавої комп’ютерної гри, що призначена для розважальних цілей.

В результаті розрахунків, виконаних у економічному розділі, визначено, що в загальній складності необхідно 592 людино-годин для розробки даного програмного забезпечення, очікувана тривалість розробки складе 3,4 місяця, а витрати на створення даного програмного забезпечення складатимуть 60619 грн.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Аналіз популярності веб сайтів - URL: http://compare.easycounter.com/ – Дата доступу : 27.04.2021.

2. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання : (ГОСТ 7.1-2003, IDT) : ДСТУ ГОСТ 7.1:2006. – Чинний з 2007–07–01. – К. : Держспоживстандарт України, 2007. – 47 с. – (Система стандартів з інформації, бібліотечної та видавничої справи) (Національний стандарт України).

3. Бусигін Б.С., Коротенко Г.М., Коротенко Л.М. Прикладна інформатика. Підручник для студентів комп'ютерних спеціальностей. – Дніпропетровськ: Видавництво НГУ, 2004. – 559 с. URL: http://www.programmer.dp.ua/book-ua-k01.php. дата звернення: 15.03.2019.

4. Бусыгин Б.С., Дивизинюк М.М., Коротенко Г.М., Коротенко Л.М. Введение в современную информатику. Учебник. – Севастополь: Издательство СНУЯЭиП, 2005. – 644 с. / URL: http://www.programmer.dp.ua/book-ru-k02.php. дата звернення: 15.01.2018.

5. Документація Unity URL: http://docs.unity3d.com/Manual/ UnityManual.html/. дата звернення: 15.01.2021.

6. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення : ДСТУ 3008-95. – Чинний від 1996–01–01. – К. : Держстандарт України, 1996. – 39 с.

7. ДСТУ 2394-94 Інформація та документація. Комплектування фонду, бібліографічний опис, аналіз документів. Терміни та визначення. – Чинний від 01.01.1995. - Київ : Держстандарт України, 1994. – 88 с.

8. Методичні вказівки з виконання економічного розділу в дипломних проектах студентів спеціальності “Комп’ютерні системи ” / Укладачі О.Г. Вагонова, Нікітіна О.Б. Н.Н. Романюк – Дніпропетровськ: Національний гірничий університет. – 2013. – 23с.

9. Методичні рекомендації до виконання кваліфікаційних робіт бакалаврів напряму підготовки 122 «Комп’ютерні науки» галузі знань 12 Інформаційні технології/, Л.М. Коротенко , О.С. Шевцова; Нац. гірн. ун-т. – Д : ДВНЗ НГУ, 2019. – 65 с.

10. Офіційний сайт середовища розробки Visual Studio Code URL: документація - URL : https://code.visualstudio.com/docs Дата доступу: 27.04.2021.

11. Скорочення слів в українській мові у бібліографічному описі. Загальні правила та вимоги : ДСТУ 3582-97. – Чинний від 1998–07–01. – К. : Держстандарт України, 1998. – 24 с. – (Державний стандарт України).

12. Стаття «Жанри відеоігор» URL: https://uk.wikipedia.org/wiki/Жанри_відеоігор Дата доступу : 27.04.2021.

13. Стаття «Мелкая моторика рук или влияние действий рук на развитие головного мозга» URL: http://erudite.com.ua/dir.php?id=874 Дата доступу : 27.04.2021.

14. Стаття «Раннер – жанр гри» URL: https://pikabu.ru/story/ chto_za_zhanr_endless_infinity_running_game_5778038 Дата доступу : 27.04.2021.

15. Стаття «C Sharp» URL:https://uk.wikipedia.org/wiki/ C_Sharp Дата доступу : 27.04.2021.

16. Стаття «Історія відеоігор» URL: https://uk.wikipedia.org/wiki/ Історія_відеоігор Дата доступу : 27.04.2021.

17. Офіційний сайт Unity - URL: http://unity3d.com/ru/ – Дата доступу : 27.04.2021.

18. Популярний портал розробників ігор- URL: http://www.gamedev.net/ – Дата доступу : 27.04.2021

19. Шилдт Г. C# 4.0: повне керівництво / Г. Шилдт., 2011. – 451 с.

ДОДАТОК А

КОД ПРОГРАМИ

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
public class BoardHighlights : MonoBehaviour {
public static BoardHighlights Instance { get; set; }
public GameObject selectedPref;
public GameObject highlightPrefab;
public GameObject hoverHighlightPrefab;
public GameObject killerHighlighPrefab;
private List<GameObject> highlights;
private GameObject parent;
private GameObject selectedHighlight;
private GameObject hoverHighlight;
private GameObject killerHighlight;
private GameObject checkedHighlight;
private void Start()

{
Instance = this;
highlights = new List<GameObject>();
parent = new GameObject("Highlights");
selectedHighlight = Instantiate(selectedPref); selectedHighlight.SetActive(false);
hoverHighlight = Instantiate(hoverHighlightPrefab); hoverHighlight.SetActive(false);
killerHighlight = Instantiate(killerHighlighPrefab); killerHighlight.SetActive(false);
checkedHighlight = Instantiate(killerHighlighPrefab); checkedHighlight.SetActive(false);
}

public GameObject GetHighlightObject()
{
GameObject go = highlights.Find(g => !g.activeSelf);
if (go == null)
{
go = Instantiate(highlightPrefab);
go.transform.parent = parent.transform;
highlights.Add(go);
}
return go;
}
public void HighlightAllowedMoves(bool[,] moves)
{
for (int i = 0; i < 8; i++)
{
for (int j = 0; j < 8; j++)
{
if (moves[i, j])


{
```

```
GameObject go = GetHighlightObject();
go.SetActive(true);
go.transform.position = new Vector3(i + 0.5f, 0, j + 0.5f);
}
}
}
}
public void HideHighlights()
{
foreach (GameObject go in highlights)
go.SetActive(false);
selectedHighlight.SetActive(false);
}
public void HighlightSelected(Vector3 position)
{
selectedHighlight.transform.position = position; selectedHighlight.SetActive(true);
}
public void ShowHoverHighlight(Vector3 pos)
{
hoverHighlight.transform.position = pos;
hoverHighlight.SetActive(true);
}
public void HideHoverHighlight()
{
hoverHighlight.SetActive(false);
}
public void ShowKillerHighlight(Vector3 pos)
{
killerHighlight.transform.position = pos;
killerHighlight.SetActive(true);
}
public void HideKillerHighlightAfter(float seconds)
{
Invoke("HideKillerHighlight", seconds);
}
public void HideKillerHighlight()
{
killerHighlight.SetActive(false);
}
public void ShowCheckedHighlight(Vector3 pos)
{
checkedHighlight.transform.position = pos;
checkedHighlight.SetActive(true);
}
public void HideCheckedHighlight()
{
checkedHighlight.SetActive(false);
}
}

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

```
using System;
public class BoardManager : MonoBehaviour
{
private const float DELAY_TIME = 1.5F;
private const float ROTATE_TIME = 0.1F;
private const float TILE_SIZE = 1.0f;
private const float TILE_OFFSET = 0.5f;
private int selectionX = -1;
private int selectionY = -1;
public List<GameObject> ChessmanPrefabs;
private List<GameObject> activeChessmans;
public Chessman[,] Chessmans { get; set; }
private Chessman selectedChessman;
private bool isWhiteTurn = true;
public static BoardManager Instance { get; set; } private bool[,] allowedMoves { get; set; }
public int[] EnPassantMove { set; get; }
public ButtonManager buttonManager;
public AudioClip cellHoverAudio;
public AudioClip pieceSelectedAudio;
private void Start()
{
Instance = this;
Chessmans = new Chessman[8, 8];
EnPassantMove = new int[2] { -1, -1 };
SpawnAllChessmans();
ShowPowerEffectAllChessmans();
GameObject.Find("MainGameManager").GetComponent<MainGameManager>().HideWin
Text();
}
private void ShowPowerEffectAllChessmans()
{
foreach (GameObject piece in activeChessmans)
{
piece.GetComponent<Chessman>().PlayPowerEffectFor(5.0f);
}
}
private void Update()
{
UpdateSelection();
DrawChessBoard();
HighLightMouseHoverCell();
if (Input.GetMouseButtonDown(0))
{
if (selectionX >= 0 && selectionY >= 0)
{
if (GameManager.Instance.GameMode == GameManager.MODE.PLAYER_VS_PLAYER)
{
if (selectedChessman == null)
{
//        select the chessman SelectChessman(selectionX, selectionY);
}
else
```

```
        {
        if (Chessmans[selectionX, selectionY] && Chess-mans[selectionX,selectionY].isWhite ==
isWhiteTurn) //The same team, reselect
        {
        SelectChessman(selectionX, selectionY);
        }
        else
        {
        //        move the chessman MoveChessman(selectionX, selectionY);
        }
        }
        }
        }
        }
        }
        private void PlayCellHoverSound()
        {
        GetComponent<AudioSource>().PlayOneShot(cellHoverAudio);
        }
        private void PlayPieceSelectedSound()
        {
        GetComponent<AudioSource>().PlayOneShot(pieceSelectedAudio);
        }
        private int oldSelectionX, oldSelectionY;
        private void HighLightMouseHoverCell()
        {
        if (GameManager.Instance.GameMode != GameManager.MODE.VISUALIZE)
        {
        if (selectionX != -1 && selectionY != -1)
        {
        if (oldSelectionX != selectionX || oldSelectionY != selectionY)
        {
        BoardHighlights.Instance.ShowHoverHighlight(new Vector3(selectionX + 0.5f, 0,
selectionY + 0.5f));
        PlayCellHoverSound();
        oldSelectionX = selectionX;
        oldSelectionY = selectionY;
        }
        }
        else
        {
        BoardHighlights.Instance.HideHoverHighlight();
        }
        }
        }
        private void UpdateSelection()
        {
        if (!Camera.main)
        return;
        RaycastHit hit;
        if (Physics.Raycast(
        Camera.main.ScreenPointToRay(Input.mousePosition),
```

```
out hit, 25f,
LayerMask.GetMask("ClickMask")))
{
selectionX = -1;
selectionY = -1;
//Debug.Log("Click on mask");
}
else if (Physics.Raycast(
Camera.main.ScreenPointToRay(Input.mousePosition),
out hit, 25f,
LayerMask.GetMask("ChessPlan")))
{
selectionX = (int)hit.point.x;
selectionY = (int)hit.point.z;
//Debug.Log("Click on Chessboard");
} else {
selectionX = -1;
selectionY = -1;
}
}
private void DrawChessBoard()
{
Vector3 widthLine = Vector3.right * 8;
Vector3 heightLine = Vector3.forward * 8;
for (int i = 0; i <= 8; i++)
{
Vector3 start = Vector3.forward * i;
Debug.DrawLine(start, start + widthLine);
for (int j = 0; j <= 8; j++)
{
start = Vector3.right * j;
Debug.DrawLine(start, start + heightLine);
}
}
// Draw selection
if (selectionX >= 0 && selectionY >= 0)
{
Debug.DrawLine(Vector3.forward * selectionY + Vector3.right * selectionX,
Vector3.forward * (selectionY + 1) + Vector3.right * (selectionX + 1));
Debug.DrawLine(Vector3.forward * (selectionY + 1) + Vector3.right * selec-
tionX,
Vector3.forward * selectionY + Vector3.right * (selectionX + 1));
}
}
public List<GameObject> GetAllChessmans()
{
return activeChessmans;
}
private void SpawnChessman(int index, int x, int y)
{
GameObject obj = Instantiate(
ChessmanPrefabs[index],
```

```
GetTileCenter(x, y),
ChessmanPrefabs[index].transform.rotation) as GameObject;
obj.transform.SetParent(this.transform); Chessmans[x, y] =
obj.GetComponent<Chessman>(); Chessmans[x, y].SetPosition(x, y); activeChessmans.Add(obj);
}
private Vector3 GetTileCenter(int x, int z)
{
Vector3 origin = new Vector3();
origin.x = TILE_SIZE * x + TILE_OFFSET;
origin.z = TILE_SIZE * z + TILE_OFFSET;
return origin;
}
//private Vector3 AdjustChessmanPosition(Vector3 pos)
//{
//	return pos + new Vector3(dx, dy, dz);
//}
private void SpawnAllChessmans()
{
activeChessmans = new List<GameObject>();
//	White team
//	King SpawnChessman(0, 4, 0);
//	Queen SpawnChessman(1, 3, 0);
//	Rooks SpawnChessman(2, 0, 0); SpawnChessman(2, 7, 0);
//	Bishops SpawnChessman(3, 2, 0); SpawnChessman(3, 5, 0);
//	Knights SpawnChessman(4, 1, 0); SpawnChessman(4, 6, 0);
//	Pawns
for (int i = 0; i < 8; i++)
SpawnChessman(5, i, 1);
//	Black team
//	King SpawnChessman(6, 4, 7);
//	Queen SpawnChessman(7, 3, 7);
//	Rooks SpawnChessman(8, 0, 7); SpawnChessman(8, 7, 7);
//	Bishops SpawnChessman(9, 2, 7); SpawnChessman(9, 5, 7);
//	Knights SpawnChessman(10, 1, 7); SpawnChessman(10, 6, 7);
//	Pawns
for (int i = 0; i < 8; i++)
SpawnChessman(11, i, 6);
}
public void SelectChessman(int x, int y)
{
if (Chessmans[x, y] == null)
return;
if (Chessmans[x, y].isWhite != isWhiteTurn)
return;
PlayPieceSelectedSound();
bool hasAtLeastOneMove = false;
allowedMoves = Chessmans[x, y].PossibleMove();
for (int i = 0; i < 8; i++)
for (int j = 0; j < 8; j++)
if (allowedMoves[i, j])
{
hasAtLeastOneMove = true;
```

```
        break;
        }
        if (!hasAtLeastOneMove)
        return;
        selectedChessman = Chessmans[x, y];
        //      change chessman material
        //previousMat = selectedChess-
        man.GetComponentInChildren<MeshRenderer>().material; //selectedMat.mainTexture =
previousMat.mainTexture;
//selectedChessman.GetComponentInChildren<MeshRenderer>().material = selected-
        Mat;
        BoardHighlights.Instance.HideHighlights();
BoardHighlights.Instance.HighlightAllowedMoves(allowedMoves);
BoardHighlights.Instance.HighlightSelected(new Vector3(x + 0.5f, 0, y + 0.5f));
        }
        public void MoveChessman(int x, int y)
        {
        if (allowedMoves[x, y])
        {
        if (chessMate != null)
        {
        BoardHighlights.Instance.HideCheckedHighlight();
        chessMate.HidePowerEffect();
        chessMate = null;
        }
        Chessman c = Chessmans[x, y];
        float delays = 0f;
        //      Check EnPassantMove ProcessEnPassantMove(c, x, y, out delays);
        //      Eat chessman
        bool isEatChess = false;
        if (c != null && c.isWhite != isWhiteTurn)
        {
        delays = DELAY_TIME;
        c.RotateEach(ROTATE_TIME);
        c.DestroyAfter(delays);
        BoardHighlights.Instance.ShowKillerHighlight(new Vector3(x + 0.5f, 0, y+ 0.5f));
        BoardHighlights.Instance.HideKillerHighlightAfter(delays); isEatChess = true;
        if (c.GetType() == typeof(King))
        {
        EndGame();
        return;
        }
        }
        BoardHighlights.Instance.ShowHoverHighlight(new Vector3(x+0.5f, 0, y+0.5f));
        //      check if pawn step on final line ProcessIfPawnStepOnFinalLine(x, y);
        //      Move selected chessman to x, y position MoveSelectedChessmanTo(x, y, delays);
        //      Checkmate
        if (!IsCheckmate(Chessmans[x, y].PossibleMove()) && isEatChess)
        {
        selectedChessman.PlayPowerEffectFor(DELAY_TIME);
        }
        ProcessCheckmate(x, y);
```

```
        // Change turn
        isWhiteTurn = !isWhiteTurn;
        }
        BoardHighlights.Instance.HideHighlights();
        selectedChessman = null;
        }
        private void ProcessIfPawnStepOnFinalLine(int x, int y)
        {
        if (selectedChessman.GetType() == typeof(Pawn))
        {
        //      check if pawn steps on final line, it becomes Queen if (y == 7) // white team
        {
        int currentX = selectedChessman.CurrentX, currentY = selectedChess-
        man.CurrentY;
        //      remove selected chessman
activeChessmans.Remove(selectedChessman.gameObject);
Destroy(selectedChessman.gameObject);
        //      spawn new chessman
        SpawnChessman(1, currentX, currentY); selectedChessman = Chessmans[currentX,
currentY];
        //      rotate the chessman selectedChessman.RotateEach(ROTATE_TIME);
        }
        else if (y == 0) // black team
        {
        int currentX = selectedChessman.CurrentX, currentY = selectedChess-
        man.CurrentY;
        //      remove selected chessman
activeChessmans.Remove(selectedChessman.gameObject);
Destroy(selectedChessman.gameObject);
        //      spawn new chessman
        SpawnChessman(7, currentX, currentY); selectedChessman = Chessmans[currentX,
currentY];
        //      rotate the chessman
        //      selectedChessman.RotateEach(ROTATE_TIME);
        }
        }
        }
        private void MoveSelectedChessmanTo(int x, int y, float delays)
        {
        Chessmans[selectedChessman.CurrentX, selectedChessman.CurrentY] = null;
        selectedChessman.MoveAfter(delays, GetTileCenter(x, y));
        selectedChessman.SetPosition(x, y);
        Chessmans[x, y] = selectedChessman;
        }
        private Chessman chessMate;
        private void ProcessCheckmate(int x, int y)
        {
        bool[,] allowedMoves = Chessmans[x, y].PossibleMove(); if (IsCheckmate(allowedMoves))
        {
        Chessman kingPos = GetKingPos(!isWhiteTurn);
BoardHighlights.Instance.ShowCheckedHighlight(new Vector3(kingPos.CurrentX +0.5f, 0,
kingPos.CurrentY + 0.5f)); selectedChessman.ShowPowerEffect(); chessMate = selectedChessman;
```

```
OnChecked();
        //if (isWhiteTurn)
        //      Debug.Log("Black team is checkmated");
        //else
        //      Debug.Log("White team is checkmated");
        }
        }
        private void OnChecked()
        {
        GameObject mainGameMangerGO = GameObject.Find("MainGameManager"); if
(mainGameMangerGO != null)
        {
        mainGameMangerGO.GetComponent<MainGameManager>().OnChecked();
        }
        }
        private void ProcessEnPassantMove(Chessman c, int x, int y, out float delay) {
        delay = 0;
        if (x == EnPassantMove[0] && y == EnPassantMove[1])
        {
        if (isWhiteTurn)
        c = Chessmans[x, y - 1];
        else
        c = Chessmans[x, y + 1];
        c.RotateEach(ROTATE_TIME);
        c.DestroyAfter(DELAY_TIME);
        //selectedChessman.RotateEach(ROTATE_TIME);
        delay = DELAY_TIME;
        }
        EnPassantMove[0] = -1;
        EnPassantMove[1] = -1;
        // EnPassant
        if (selectedChessman.GetType() == typeof(Pawn))
        {
        if (selectedChessman.CurrentY == 1 && y == 3)
        {
        EnPassantMove[0] = x;
        EnPassantMove[1] = y - 1;
        }
        else if (selectedChessman.CurrentY == 6 && y == 4)
        {
        EnPassantMove[0] = x;
        EnPassantMove[1] = y + 1;
        }
        }
        }
        private Chessman GetKingPos(bool isWhite)
        {
        for (int i = 0; i < 8; i++)
        {
        for (int j = 0; j < 8; j++)
        {
        if ( Chessmans[i, j] != null && Chessmans[i, j].GetType() ==
```

```
typeof(King)
&& Chessmans[i, j].isWhite == isWhite)
{
return Chessmans[i,j];
}
}
}
return null;
}
private bool IsCheckmate(bool[,] allowedMoves)
{
if (allowedMoves.Length == 0)
return false;
for (int i = 0; i < 8; i++)
{
for (int j = 0; j < 8; j++)
{
if (allowedMoves[i, j] && Chessmans[i, j] != null && Chessmans[i, j].GetType() ==
typeof(King))
{
return true;
}
}
}
return false;
}
private void EndGame()
{
if (isWhiteTurn)
GameObject.Find("MainGameManager").GetComponent<MainGameManager>().EndGame
(1); else
GameObject.Find("MainGameManager").GetComponent<MainGameManager>().EndGame
(-1);
foreach (GameObject go in activeChessmans)
{
Destroy(go);
}
BoardHighlights.Instance.HideHighlights();
SpawnAllChessmans();
}
public Location Find(int turn, char c, Location dst, string disam)
{
Location lo = new Location();
bool isWhite = (turn == 0);
foreach (GameObject chessObj in activeChessmans)
{
Chessman chess = chessObj.GetComponent<Chessman>();
if ((chess.isWhite == isWhite) && (chess.Annotation().Equals(c.ToString())))
{
if (chess.CanGo(dst.x, dst.y))
{
if (disam.Length == 1) //Disambiguating moves
```

```
{
if (disam[0] >= '1' && disam[0] <= '9') //rank have to the same {
if (chess.CurrentY == (disam[0] - '1'))
{
return new Location(chess.CurrentX, chess.CurrentY);
}
}
else if (disam[0] >= 'a' && disam[0] <= 'z') //file have to thesame
{
if (chess.CurrentX == (disam[0] - 'a'))
{
return new Location(chess.CurrentX, chess.CurrentY);
}
}
else
{
Debug.Log("Unexpected result! " + disam);
}
}
else
{
return new Location(chess.CurrentX, chess.CurrentY);
}
}
}
}
return lo;
}
public void KingSideCastling(int turn)
{
bool isWhite = (turn == 0);
int rank = isWhite ? 0 : 7;
if (Chessmans[7, rank] && Chessmans[7, rank].Annotation().Equals("R")
&& Chessmans[4, rank] && Chessmans[4, rank].Annotation().Equals("K"))
{
MoveFromTo(7, rank, 5, rank);
MoveFromTo(4, rank, 6, rank);
Chessmans[5, rank].PlayPowerEffectFor(2.0f);
Chessmans[6, rank].PlayPowerEffectFor(2.0f);
}
}
private GameObject GetChessmanObj(int x, int y)
{
foreach (GameObject pieceObj in activeChessmans)
{
Chessman piece = pieceObj.GetComponent<Chessman>(); if (piece.CurrentX == x &&
piece.CurrentY == y)
return pieceObj;
}
return null;
}
private void MoveFromTo(int srcX, int srcY, int dstX, int dstY)
```

```
        {
        Chessman chess = Chessmans[srcX, srcY];
        Chessmans[srcX, srcY] = null; chess.transform.position = GetTileCenter(dstX, dstY);
chess.SetPosition(dstX, dstY);
        Chessmans[dstX, dstY] = chess;
        }
        public void QueenSideCastling(int turn)
        {
        bool isWhite = (turn == 0);
        int rank = isWhite ? 0 : 7;
        if (Chessmans[0, rank] && Chessmans[0, rank].Annotation().Equals("R")
        && Chessmans[4, rank] && Chessmans[4, rank].Annotation().Equals("K"))
        {
        MoveFromTo(0, rank, 3, rank);
        MoveFromTo(4, rank, 2, rank);
        }
        }
        }

        using UnityEngine;
        using System.Collections;
        public class ButtonManager : MonoBehaviour
        {
        Animator cameraAnimator;
        int index = 0;
        private string[] cameraTrigger = { "WhiteTurn", "RightMotion", "BlackTurn", "LeftMo-
tion"};
        Animator pauseMenuAnimator;
        GameObject clickMash;
        bool isShowPauseMenu = false;
        //        Use this for initialization void Start()
        {
        cameraAnimator = GameObject.Find("Main Camera").GetComponent<Animator>(); if
(cameraAnimator == null)
        {
        Debug.Log("Can't find camera animator.");
        }
        pauseMenuAnimator = GameObject.Find("Pause Menu").GetComponent<Animator>();
        if (pauseMenuAnimator == null)
        {
        Debug.Log("Can't find pause menu animator.");
        }
        clickMash = GameObject.Find("ClickMask");
        if (clickMash == null)
        {
        Debug.Log("Can't find ClickMask.");
        } else
        {
        clickMash.SetActive(false);
        }
        }
        void Update()
```

```
{
if (Input.GetKeyDown(KeyCode.Escape))
{
if (!isShowPauseMenu)
{
pauseMenuAnimator.SetTrigger("ShowPauseMenu"); isShowPauseMenu = true;
clickMash.SetActive(true);
} else
{
pauseMenuAnimator.SetTrigger("HidePauseMenu"); isShowPauseMenu = false;
clickMash.SetActive(false);
}
}
}
// side: left, right, white, black
public void MoveCamera(string side)
{
if (side == "left")
{
index--;
if (index < 0)
index = 3;
} else
{
index++;
if (index > 3) index = 0;
}
cameraAnimator.SetTrigger(cameraTrigger[index]);
}
public void ExitGame()
{
LevelManager.QuitRequest();
}
public void LoadMainMenu()
{
LevelManager.LoadLevel("MainMenu");
}
}

using UnityEngine;
using System.Collections;
public class CameraController : MonoBehaviour {
//	Use this for initialization void Start () {
transform.RotateAround(new Vector3(4, 0, 4), Vector3.right, 90);
}
//	Update is called once per frame
void Update () {
}
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
public class GameManager {
private static GameManager _instance = null;
public static GameManager Instance
{
get
{
if (_instance == null)
_instance = new GameManager();
return _instance;
}
}
public enum MODE { PLAYER_VS_PLAYER, VISUALIZE, PLAYER_VS_AI }
public MODE GameMode = MODE.PLAYER_VS_PLAYER;
public string VisualizePath = null;
public void StartGame()
{
LevelManager.LoadLevel("MainGame");
}
}

using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;
public class LevelManager : MonoBehaviour {
public static void LoadLevel(string name) {
//Debug.Log
("Level load requested for: " + name);
//Application.LoadLevel (name);
SceneManager.LoadScene(name);
}
public static void QuitRequest(){
//Debug.Log
("I want to quit!");
Applica-
tion.Quit ();
}
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public struct Location {
public int x;
public int y;
public Location(int x, int y)
{
this.x = x;
this.y = y;
}
public override string ToString()
{
return x + "," + y;
```

```
}
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class MainGameBGM : MonoBehaviour {
public Animator animator;
private bool isFirst = true;
public AudioClip preAudio;
public AudioClip backAudio;
//      Use this for initialization void Start () {
GetComponent<AudioSource>().clip = preAudio; GetComponent<AudioSource>().Play();
}
//      Update is called once per frame
void Update () {
if (isFirst)
{
if (animator.IsInTransition(0))
{
GetComponent<AudioSource>().clip = backAudio;
GetComponent<AudioSource>().Play();
isFirst = false;
}
}
}
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class VisualizeItem : MonoBehaviour {
public string filePath;
public string fileName;
//      Use this for initialization void Start () {
}
//      Update is called once per frame void Update () {
}
public void StartVisualize()
{
Debug.Log("StartVisualize" + filePath); GameManager.Instance.GameMode =
GameManager.MODE.VISUALIZE; GameManager.Instance.VisualizePath = filePath;
GameManager.Instance.StartGame();
}
}

using System.Collections;
using System.Collections.Generic;
using System.IO;
using UnityEngine;
using UnityEngine.UI;
public class VisualizeList : MonoBehaviour {
```

```
public GameObject itemGO;
public GameObject listGO;
public ScrollRect scroll;
//        Use this for initialization void Start () {
Debug.Log("VisualizeList!"); string path = "Data/";
var info = new DirectoryInfo(path); var fileInfo = info.GetFiles(); foreach (var file in
fileInfo)
{
GameObject item = (GameObject)Instantiate(itemGO, listGO.transform);
item.transform.localScale = new Vector3(1, 1, 1); item.transform.localPosition = new
Vector3(0, 0, 0); item.GetComponentInChildren<Text>().text = file.Name;
item.GetComponent<VisualizeItem>().filePath = file.DirectoryName + "\\" +
file.Name;
item.GetComponent<VisualizeItem>().fileName = file.Name;
}
scroll.verticalNormalizedPosition = 1; using System; using System.Collections;

using System.Collections.Generic;
using System.IO;
using System.Security.Cryptography;
using System.Text;
using UnityEngine;
public class VisualizeMatch : MonoBehaviour {
List<String> data;
int step = 0;
int result = 0;
MainGameManager gameManager;
public VisualizeMatch(MainGameManager gameManager)
{
this.gameManager = gameManager;
}
public void LoadMatchData()
{
data = new List<string>();
string fileName = GameManager.Instance.VisualizePath;
try
{
string line;
StreamReader theReader = new StreamReader(fileName, Encoding.Default);
do
{
line = theReader.ReadLine();
if (line != null)
{
Debug.Log(line);
if (line.Length > 0 && line[0] != '1')
continue;
String[] parts = line.Split(' ');
foreach (String part in parts)
{
if (!part.Contains("."))
data.Add(part);
```

```
}
String strResult = data[data.Count - 1];
data.RemoveAt(data.Count - 1);
if (strResult.Equals("0-1"))
result = -1; //Black win
else if (strResult.Equals("1-0"))
result = 1; //White win
else
result = 0; //Tie
}
} while (line != null); theReader.Close();
}
catch (Exception e)
{
Debug.Log("{0}\n" + e.Message);
}
}
public void Visualize()
{
for (int i = 0; i < data.Count; i++)
{
Move(i%2, data[i]);
}
}
public void VisualizeNextStep()
{
if (step == data.Count)
{
Debug.Log("End of game!");
Debug.Log("Result: " + result);
gameManager.isVisualize = false;
gameManager.EndGame(result);
return;
}
Move(step % 2, data[step]);
step++;
}
public String Normalize(String move)
{
return move.Replace("+", "").Replace("x", "");
}
public void Move(int turn, String move)
{
move = Normalize(move);
char f = move[0];
if (move == "O-O")
{
gameManager.KingSideCastling(turn);
}
else if (move == "O-O-O")
{
gameManager.QueenSideCastling(turn);
```

```
}
else
{
int n = move.Length;
int rank = move[n - 1] - '1'; //hang
int file = move[n - 2] - 'a'; //cot
Location dest = new Location(file, rank);
Location src;
if ("KQBNR".Contains(f.ToString())) {
Stringdisam = "";
if (n== 4)
disam += move[1];
src =gameManager.Find(turn, f, dest, disam);
} else { //Pawn
String disam = "";
if (n== 3)
disam += move[0];
src =gameManager.Find(turn, 'P', dest, disam); //Find Pawn
}
gameManager.Move(src, dest);
}
}
}
//      Update is called once per frame void Update () {
}
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class CustomCursor : MonoBehaviour {
[SerializeField]
Texture2D yourCursor;
[SerializeField]
int cursorSizeX;
[SerializeField]
int cursorSizeY;
void Awake()
{
DontDestroyOnLoad(transform.gameObject);
}
//      Use this for initialization void Start()
{
Cursor.visible = false;
}
//      Update is called once per frame void Update()
{
}
void OnGUI()
{
GUI.DrawTexture(new Rect(Event.current.mousePosition.x, Event.current.mousePosition.y,
cursorSizeX, cursorSizeY), yourCursor);
```

```
}
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;
public class MenuItemHandler : MonoBehaviour, IPointerEnterHandler {
[SerializeField]
AudioClip soundEffect;
public void OnPointerEnter(PointerEventData eventData)
{
GetComponent<AudioSource>().PlayOneShot(soundEffect);
}
}

using UnityEngine;
using System.Collections;
using System;
public class Bishop : Chessman {
public override string Annotation()
{
return "B";
}
public override bool[,] PossibleEat()
{
return PossibleMove();
}
public override bool[,] PossibleMove()
{
bool[,] moves = new bool[8, 8];
Chessman c;
int i, j;
//        Top Left i = CurrentX; j = CurrentY; while (true)
{
i--; j++;
if (i < 0 || j > 7) break;
c = BoardManager.Instance.Chessmans[i, j];
if (c == null)
moves[i, j] = true;
else if (c.isWhite != isWhite)
{
moves[i, j] = true;
break;
}
else
break;
}
//        Top Right i = CurrentX; j = CurrentY; while (true)
{
i++; j++;
```

```
if (i > 7 || j > 7) break;
c = BoardManager.Instance.Chessmans[i, j];
if (c == null)
moves[i, j] = true;
else if (c.isWhite != isWhite)
{
moves[i, j] = true;
break;
}
else
break;
}
//        Bottom Left i = CurrentX; j = CurrentY; while (true)
{
i--; j--;
if (i < 0 || j < 0) break;
c = BoardManager.Instance.Chessmans[i, j];
if (c == null)
moves[i, j] = true;
else if (c.isWhite != isWhite)
{
moves[i, j] = true;
break;
}
else
break;
}
//        Bottom Right i = CurrentX; j = CurrentY; while (true)
{
i++; j--;
if (i > 7 || j < 0) break;
c = BoardManager.Instance.Chessmans[i, j];
if (c == null)
moves[i, j] = true;
else if (c.isWhite != isWhite)
{
moves[i, j] = true;
break;
}
else
break;
}
return moves;
}
}

using UnityEngine;
using System.Collections;
using System;
public abstract class Chessman : MonoBehaviour { public int CurrentX { get; set; }
public int CurrentY { get; set; }
public bool isWhite;
```

```
        public abstract string Annotation();
        private Quaternion originRotation;
        private float speed = 1f;
        private float incSpeed = 1.0f;
        private Animator anim;
        private bool startMove = false;
        private AudioSource _audioSource = null;
        private AudioSource AudioSource
        {
        get
        {
        if (_audioSource == null)
        {
        this.gameObject.AddComponent<AudioSource>(); return
this.gameObject.GetComponent<AudioSource>();
        }
        return _audioSource;
        }
        }
        private static AudioClip _manDeadAudio;
        private static AudioClip ManDeadAudio
        {
        get {
        if (_manDeadAudio == null)
        {
        _manDeadAudio = Resources.Load<AudioClip>("Sound/Game/dead_man");
        }
        return _manDeadAudio;
        }
        }
        private static AudioClip _femaleDeadAudio;
        private static AudioClip FemaleDeadAudio
        {
        get
        {
        if (_femaleDeadAudio == null)
        {
        _femaleDeadAudio = Resources.Load<AudioClip>("Sound/Game/dead_female");
        }
        return _femaleDeadAudio;
        }
        }
        private static AudioClip _moveSurfAudio;
        private static AudioClip MoveSurfAudio
        {
        get
        {
        if (_moveSurfAudio == null)
        {
        _moveSurfAudio = Resources.Load<AudioClip>("Sound/Game/move_surf");
        }
        return _moveSurfAudio;
```

```
}
}
private static GameObject _VFX_ParticlePath;
private static GameObject VFX_ParticlePath
{
get
{
if (_VFX_ParticlePath == null)
{
_VFX_ParticlePath = Re-
sources.Load<GameObject>("Prefabs/VFX/VFX_ParticlePath");
}
return _VFX_ParticlePath;
}
}
private GameObject _particlePathGO = null;
private GameObject ParticlePathGO
{
get
{
if (_particlePathGO == null)
{
_particlePathGO = Instantiate(VFX_ParticlePath);
_particlePathGO.transform.parent = gameObject.transform;
if (isWhite)
{
_particlePathGO.transform.localPosition = new Vector3(0, 0.5f, -
0.2f);
} else
{
_particlePathGO.transform.localPosition = new Vector3(0, 0.5f,0.2f);
}
}
return _particlePathGO;
}
}
private void Start()
{
originRotation = transform.rotation;
//Animator[] anims = GetComponentsInChildren<Animator>();
//anim = anims[0];
//if (anims.Length >= 2)
//        anim = anims[anims.Length - 1];
//if (anim == null)
//{
//        Debug.Log("Chessman's animator is null");
//}
}
private void Update()
{
if (startMove)
{
```

```
float step = speed * Time.deltaTime;
transform.position = Vector3.MoveTowards(transform.position, newPosition,step);
speed += incSpeed;
if (transform.position == newPosition)
{
newPosition = Vector3.zero;
startMove = false;
speed = 1f;
HideMoveEffect();
}
}
}
private void HideMoveEffect()
{
ParticlePathGO.SetActive(false);
}
private void ShowMoveEffect()
{
ParticlePathGO.SetActive(true);
}
public void SetPosition(int x, int y)
{
CurrentX = x;
CurrentY = y;
}
public virtual bool[,] PossibleMove()
{
return new bool[8, 8];
}
public virtual bool[,] PossibleEat()
{
return new bool[8, 8];
}
public bool CanGo(int x, int y)
{
bool[,] possible = this.PossibleMove();
return possible[x, y];
}
internal void RotateEach(float seconds)
{
InvokeRepeating("Rotate", 0f, seconds);
}
private GameObject _powerEffect;
private GameObject PowerEffect
{
get
{
if (_powerEffect == null)
{
_powerEffect = gameObject.transform.GetChild(2).gameObject;
}
return _powerEffect;
```

```
}
}
public void PlayPowerEffectFor(float seconds)
{
if (PowerEffect == null)
{
Debug.Log(CurrentX + ";" + CurrentY + " " + Annotation());
}
PowerEffect.SetActive(true);
Invoke("HidePowerEffect", seconds);
}
public void ShowPowerEffect()
{
PowerEffect.SetActive(true);
}
public void HidePowerEffect()
{
PowerEffect.SetActive(false);
}
private int t = 5;
private int dt = 5;
private void Rotate()
{
transform.Rotate(Vector3.up * t);
t += dt;
}
internal void DestroyAfter(float seconds)
{
PlayDieSound();
Invoke("DestroyGameObject", seconds);
}
private void PlayDieSound()
{
if (this.Annotation() != "Q")
{
AudioSource.PlayOneShot(ManDeadAudio);
}
else
{
AudioSource.PlayOneShot(FemaleDeadAudio);
}
}
private void DestroyGameObject()
{
BoardManager.Instance.GetAllChessmans().Remove(gameObject); Destroy(gameObject);
}
private Vector3 newPosition = Vector3.zero;
internal void MoveAfter(float seconds, Vector3 position)
{
newPosition = position;
Invoke("Move", seconds);
}
```

```
private void PlayMoveSoundEffect()
{
AudioSource.PlayOneShot(MoveSurfAudio);
}
private void Move()
{
startMove = true;
ShowMoveEffect();
PlayMoveSoundEffect();
//transform.position = newPosition;
//        stop rotation
//        CancelInvoke("Rotate");
//        restore origin rotation //transform.rotation = originRotation;
//        restore rotate speed //t = 10;
}
}
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом_ doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Диплом_.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Program.rar	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація_.ppt	Презентація кваліфікаційної роботи