РЕФЕРАТ

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

Об'єкт розробки: комп’ютерна гра жанру Auto battler.

Мета кваліфікаційної роботи: реалізація стратегічної відеогри Valhalla Auto Chess на платформі Unity.

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 платформу для розробки, виконано проєктування 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 полягає у створенні програмного додатка, в якому реалізовано на основі розробленого алгоритму прийняття рішень досконалу стратегію ведення гри жанру Auto battler та запропоновано працеспроможний кінцевий продукт для його використання в розважальних цілях

Актуальність даного програмного продукту визначається великим попитом на подібні розробки, тому щов наш час є доцільним використання ігор для розваг людей. Дана програма - гра може використовуватись користувачами для розважальних заходів та приємного відпочинку.

Список ключових слів: ГРА, ЮНІТИ, КОМПОНЕНТИ, ДИЗАЙН, ПРОГРАМА, ІГРОВИЙ ПРОЦЕС, АЛГОРИТМ, ГРАФІКА, ВІЗУАЛІЗАЦІЯ.

ABSTRACT

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

Object of development: computer game of the genre Auto battler.

The purpose of the qualification work: implementation of the strategic video game Valhalla Auto Chess on the Unity platform.

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, designs and develops 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 lies in the creation of a software application, which implements on the basis of the developed decision-making algorithm a perfect strategy for the game genre Auto battler and offers a workable end product for its use for entertainment purposes.

The relevance of this software product is determined by the high demand for such developments, because nowadays it is advisable to use games to entertain people. This program - the game can be used by users for entertainment and recreation.

Keywords: GAME, UNITS, COMPONENTS, DESIGN, PROGRAM, GAME PROCESS, ALGORITHM, GRAPHICS, VISUALIZATION.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ГІК – Графічний інтерфейс користувача;

ІС – інформаційна система;

ОС – операційна система;

ПЗ – програмне забезпечення;

ПК – персональний комп’ютер.

ЗМІСТ

ВСТУП

На сьогоднішній день ринок розваг є одним з найприбутковіших. З моменту початку інформаційно-технічної революції світ стрімко рухається в майбутнє, створюючи все більш досконалі комп'ютери.

З появою персональних комп'ютерів, з кожним роком, їх роль в житті людей постійно зростає. Комп'ютер став незамінним помічником не тільки в сфері економічних розрахунків, а й став потужним центром розваг.

Комп'ютерні розваги роблять життя людини більш насиченим і, як наслідок – це потужна економічна сфера, яка приносить величезні доходи.

Тому не випадково, що особливу роль у житі сучасної людини відводять комп'ютерним іграм, перші з яких існували на самій зорі комп'ютерної техніки.

У світі існує надзвичайно велика кількість любителів комп'ютерних ігор з різним досвідом в цій сфері. Є початківці, є навчені досвідом гравці, тому на цьому ринку затребувані ігрові проекти різної спрямованості.

Багато програмістів-любителів створюють невеликі ігрові програми, які не володіють сучасною дорогою висококласною графікою і звуковим супроводом, як ігри популярних компаній, але вони часто або несуть якість інновації в гемплеї, або володіють цікавим сюжетом, і в підсумку вони так само користуються великою популярністю і приносять творцям достатні доходи.

У січні 2019 року група китайських розробників, Drodo Studio, випустила Dota Auto Chess, власну кастомну гру для гри Dota2. Популярність мода, який до травня 2019 року мав понад вісім мільйонів гравців, призвів до створення нового жанру, який випустив ряд режимів для вже створених відеоігор. На сьогоднішній день, жанр Auto battler налічує всього лише дві всесвітньо відомих гри, що робить цей жанр дуже актуальним для розробки.

Метою кваліфікаційної роботи є реалізація стратегічної відеогри Valhalla Auto Chess на платформі Unity.

Актуальність розробленої програми полягає в тому, що в наш час є доцільним використання ігор для розваг людей.

Дана програма - гра може використовуватись користувачами для розважальних заходів та приємного відпочинку.

В результаті виконання кваліфікаційної роботи створено додаток, в якому реалізовано на основі розробленого алгоритму прийняття рішень досконалу стратегію ведення гри жанру Auto battler та запропоновано працеспроможний кінцевий продукт для його використання в розважальних цілях.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

1.1.1. Основні поняття комп’ютерних ігор

Останнім часом все частіше можна почути про збільшення масштабів розробки програмного забезпечення розважального характеру. До числа таких продуктів відносяться відеоігри. Через десятки років індустрія відеоігор зайняла фіксоване місце на ринку поряд з іншими розвагами сфери мультимедіа, таких як кіно, мультиплікація, музика. Це багатомільярдні компанії, що займаються розробкою розважальних продуктів та програмних забезпечень, що складаються з мільйонів рядків програмного коду і файлів мультимедіа.

Ігри стали охоплювати величезну аудиторію по всьому світу, з'являючись на різних ігрових пристроях. Вони стали складніше і масштабніше: покращилася обробка візуального простору, обробка фізики об'єктів і штучний інтелект. Але напрямна методологія розробки не змінилася. Сучасні комп'ютерні ігри - великі і складні програмні комплекси. Витрати тільки на програмну розробку часто вимірюються сотнями людино-місяців. За обсягом задіяних технологій залучених коштів і рівня професійної підготовки розробників ігрова індустрія давно зайняла аж ніяк не останнє місце в світі IT.

Розробка апаратної бази в секторі персональних комп'ютерів вже досить довго стимулюється саме ігровою індустрією. Розвиток багатьох програмних технологій, зокрема, всіх мислимих способів відтворення тривимірних зображень, йде саме завдяки іграм.

Класифікація комп'ютерних ігор може бути наступна:

1. Пригодницькі ігри.
2. Імітатори різноманітних битв.
3. Імітатори керування пристроями

4. Спортивні імітатори.

5. Аркадні ігри/ігри дії.

6. Настільні ігри.

7. Імітатори реальності.

8. Стратегічні і військові імітатори.

9. Розвиваючі ігри для дітей.

Світ PC-ігор багато в чому схожий на світ популярної музики і фільмів: створюється багато ігор різних найменувань, деякі стають хітами і тримають популярність протягом деякого часу, інші зникають. Поступово ці хіти стає важко відшукати, і мало-помалу вони замінюються іншими кращими іграми. Дуже мало PC-ігор живуть більше 2-3 років. З тих, що дійсно залишаються, багато настільки гарні, що стають вічними, і вони заслуговують того, щоб в них грали. Деякі, як Flight Simulator або king's Quest, оновлюються час від часу, так що їх можна розглядати як практично нові ігри. Інші залишаються незмінними, і рівень продажів у них такий, що вони залишаються в обігу, але вже не рекламуються активно їх виробниками. Це ігри, які не кидаються в очі, «золоті» ігри - класичні PC-ігри.

Основними поняттями в усіх відеоіграх є наступні терміни:

1. Геймплей - ігровий процес з точки зору гравця. Геймплей включає в себе різні змістовні аспекти комп'ютерної гри, в тому числі технічні, такі як внутрішньоігрова механіка, сукупність певних методів взаємодії гри з гравцем і ін. Саме поняття геймплея вкрай узагальнено і зазвичай використовується для вираження отриманих відчуттів в ході проходження гри, під впливом таких факторів, як графіка, звук і сюжет.

2. Ігровий персонаж - позначення персонажа в комп'ютерних іграх, який управляється або може управлятися людиною-гравцем. Управління людиною відокремлює ігрові персонажі від неігрових, які управляються ігровим штучним інтелектом. У переважній більшості випадків ігровий персонаж є головним героєм і протагоністом гри.

3. Ігровий баланс – в іграх (спортивних, настільних, комп'ютерних та інших) рівновага між персонажами, командами, тактиками гри і іншими ігровими об'єктами. Ігровий баланс - одна з вимог до «чесності» правил. Особливо баланс важливий для багатокористувацьких ігор. У комп'ютерних іграх це баланс між числами, що описують різні характеристики в грі - такі, як сила пошкодження, швидкість бігу, швидкість споруди одиниць і багато іншого. Цей баланс багато в чому визначає складність, інтерес і плавність ігрового процесу. Ігровий баланс - одна з найскладніших сторін розробки ігор. Розробник гри призводить гру до стану балансу, змінюючи ігрові характеристики в ході бета-тестування. Але остаточно баланс мережевої гри відточується протягом деякого часу після виходу самої гри, так як це складний процес підгонки самих незначних відхилень від збалансованого значення.

Основною проблемою ігрових додатків, є низька кросплатформовість, є безліч дуже хороших ігор, які вимагають уваги кожного, але вони часто пишуться і розробляються тільки під одну платформу. Мета створеного продукту полягатиме в тому, щоб розробити кросплатформовий додаток.

1.1.2. Опис жанру Auto Battler

Auto Battler - жанр стратегій, в якому гравець розставляє персонажів на ігровому полі, які потім б'ються проти персонажів іншого гравця без його участі. Серед представників жанру можна згадати самостійні ігри Auto Chess і Dota Underlords, а також модифікації Teamfight Tactics і Hearthstone's Battlegrounds для ігор League of Legends і Hearthstone.

Прообразом жанру Auto Battler стала модифікація Dota Auto Chess для Dota 2 від китайських розробників з Drodo Studio, створена в січні 2019 року. У травні 2019 року в неї грало вже близько 8 мільйонів осіб.

Dota Auto Chess (2019): помітивши неймовірною успіх модифікації, великі студії (наприклад, Valve і Riot Games) створили власні ігри, які офіційно стали породженням нового жанру. Наприклад, Dota Underlords від Valve.

У кожному матчі беруть участь кілька гравців. Матч являє собою міні-турнір, в кожному раунді якого гравці випадковим чином вибудовуються в пари, в яких проводяться бої між командами персонажів гравців в автоматичному режимі. У матчі перемагає останній залишився в живих гравець.

На початку кожного раунду гравці купують юнітів, які можуть апгрейдитися в свої посилені версії, і розставляють їх на розміченому ігровому полі. Певні типи юнітів отримують бонуси при знаходженні разом на ігровому полі.

Після фази розстановки юнітів починається фаза бою. Під час неї розставлені юніти борються проти команди юнітів опонента, причому гравці не можуть впливати на перебіг і, відповідно, результат бою. Бій закінчується при знищенні всіх юнітів одного з гравців, що призводить до ураження одне з гравця в дуелі.

Після завершення бою гравець отримує шкоди, що залежить від кількості і якості залишилися в живих юнітів опонента. Якщо у гравця закінчується здоров'я, то він програє і більш не бере участі в боях цього матчу.

1.2. Призначення розробки та область застосування

Комп’ютерні ігри є невід’ємною частиною дозвілля більшості сучасних молодих людей як в Україні, так і у всьому світі

Сучасна комп’ютерна гра – це багатофункціональна програма, яку використовують не тільки з розважальними, а і з навчальними та пропагандистськими цілями.

В результаті виконання кваліфікаційної роботи буде створено додаток, в якому реалізовано на основі розробленого алгоритму прийняття рішень досконалу стратегію ведення гри «Valhalla Auto Chess» на платформі Unity та запропоновано працеспроможний кінцевий продукт для його використання в розважальних цілях.

1.3. Підстава для розробки

Підставою для розробки кваліфікаційної роботи бакалавра на тему «Розробка комп’ютерної гри жанру Auto Battler на платформі Unity» є наказ по Національному технічному університету «Дніпровська політехніка» від __.__. 2021р. № ____-_.

1.4. Постановка завдання

Метою кваліфікаційної роботи є реалізація стратегічної відеогри Valhalla Auto Chess жанру Auto Battler на платформі Unity.

Правила гри наступні: у кожному матчі беруть участь кілька гравців. Матч являє собою міні-турнір, в кожному раунді якого гравці випадковим чином вибудовуються в пари, в яких проводяться бої між командами персонажів гравців в автоматичному режимі. У матчі перемагає останній залишився в живих гравець.

На початку кожного раунду гравці купують юнітів, які можуть апгрейдитися в свої посилені версії, і розставляють їх на розміченому ігровому полі. Певні типи юнітів отримують бонуси при знаходженні разом на ігровому полі.

Після фази розстановки юнітів починається фаза бою. Під час неї розставлені юніти борються проти команди юнітів опонента, причому гравці не можуть впливати на перебіг і, відповідно, результат бою. Бій закінчується при знищенні всіх юнітів одного з гравців, що призводить до ураження одне з гравця в дуелі.

Після завершення бою гравець отримує шкоду, що залежить від кількості і якості залишилися в живих юнітів опонента. Якщо у гравця закінчується здоров'я, то він програє і більш не бере участі в боях цього матчу.

Завданням кваліфікаційної роботи є спроектувати та розробити стратегічну відео гру «Valhalla Auto Chess» на платформі Unity, метою якої є

розвинути логіку та кмітливість гравців.

Тематика даної роботи є дуже актуальною в зв’язку з активним розвитком інформаційних технологій в усіх сферах нашого життя, в тому числі в навчанні та розвагах.

Для виконання даного завдання розробити програму – комп'ютерну логічну гру. В результаті виконання даної роботи розробник засвоює прийоми практичного використання об'єктно-орієнтованого підходу до створення кінцевого програмного продукту:

- реалізує обрану комп'ютерну гру;
- володіє методами побудови графічного інтерфейсу користувача;
- задовольняє вимогу переносимості на рівні вихідного коду, простого в установці і обслуговуванні.

Для вирішення цих задач при розробці ігрового додатку повинні бути використані наступні рішення:

1. Розробка логічної схеми додатку.
2. Розробка алгоритму роботи додатку.
3. Розробка графіки додатку.
4. Розробка інтерфейсу користувача.
5. Розробка ігрового процесу.
6. Тестування розробленої програми.

У якості вхідних даних програми виступають:

– вибір фігури та направлення ходу для неї;
– вибір сторони змагання;
– вибір режиму в грі.

Вихідними даними є:

– відображення фігур;
– програвання мелодії;
– відображення анімації;
– вивід результату гри.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

До розроблюваного програмного забезпечення виставляються наступні функціональні характеристики:

1. Фігури. Переміщення фігур відбуватиметься в тривимірному просторі, можна рухатися вліво, вправо, вперед та назад.

Для кожної унікальної фігури необхідно розробити:

– свою 3D модель;

– анімації простою;

– свої текстури;

– звуки;

– унікальні атаки.

2. Поле бою. Необхідно розробити карту, яка буде відображати зміст назви гри, а саме показувати ідеальний пейзаж для вікінгів того часу. Також продумати контролер камери, щоб гравець зміг знайти для себе оптимальне положення, для більш чіткого сприйняття інформації. Ігрова дошка повинна бути розрахована для двох гравців, тобто коли відбувається хід гравця, він може взаємодіяти тільки зі своїми фігурами.

3. Графічний інтерфейс. При закінчені гри, необхідно відобразити переможця, згідно з кольору гравця. В будь-який момент гра може бути призупинена для редагування користувачем графіки та гучності звуків.

4. Головне меню міститиме кнопки виходу з гри та ігрових налаштувань. Додатково, для ознайомлення з правилами гри та особливостями додатку, в головному меню міститиметься кнопка інструкції, яка повинна запустити записаний гемплей.

1.5.2. Вимоги до інформаційної безпеки

Для надійної роботи системи необхідно:

— використовувати ліцензійне програмне забезпечення;

— здійснювати захист від несанкціонованого доступу;

— застосовувати джерело безперебійного живлення для захисту від перепадів напруги або збоїв у живленні;

— здійснювати контроль даних, що вводяться користувачем.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для запуску відеогри необхідна конфігурація ПК не гірше мінімальної (табл. 1.1), для комфортної гри не гірше рекомендованої (табл. 1.2).

Таблиця 1.1

Мінімальна конфігурація для запуску гри

Тип	Модель	Характеристики
Процесор	Intel Pentium c2400	4 ядра по 2.40GHz
Відеокарта	Intel HD Graphics 2400	CF 550, GDDR3 128-bit 1024mb
ОЗП	GR1600S3V64L11	DDR3, 2048MB, 1600 MHz
Накопичувач	WD7000AAS	320GB, 7200RPM, 8MB, 3.5

Таблиця 1.2

Рекомендована конфігурація для запуску гри

Тип	Модель	Характеристики
Процесор	Intel Core i5-2500K	4 ядра по 3.3GHz
Відеокарта	GeForce GTX750 Ti	CF 1072, GDDR5 128-bit2048mb
ОЗП	GR2400D464L17S	DDR4, 4096MB, 2400 MHz
Накопичувач	WD10EZEX	1TB, 7200RPM, 64MB, 3.5

Такий вибір компонентів є оправданим тому, що:

– процесори підтримують інструкції SSE2, необхідні для запуску;

– відеокарти підтримують версії DirectX, які використовуються;

– кількості і швидкості ОЗП вистачає для процесів;

– пам’яті ЖД вистачає для зберігання ігрових файлів.

Також для вводу даних необхідна мишка та клавіатура. Для виводу картинки потрібен монітор.

1.5.4. Вимоги до інформаційної та програмної сумісності

Для коректної роботи відеогри потрібні наступні програмні засоби:

– DirectX – це набір компонентів Windows, який дає змогу програмному забезпеченню, в основному й особливо іграм, працювати напряму з відео і аудіо устаткуванням. Він допомагає ефективніше використовувати функції мультимедійного акселератора, вбудовані в устаткування, що покращує якість відтворення мультимедійного вмісту в цілому [5]. Для запуску гри потрібна 10 версія набору;

– платформа .NET Framework – це середовище виконання, яке керує додатками, воно складається з середовища CLR, яке надає інструменти управління пам'яттю, інші служби системи, та велику бібліотеку класів, що дозволяє програмістам використовувати стійкий, надійний код у всіх основних областях розробки додатків;

– ОС Windows забезпечує комп’ютер усіма необхідними драйверами, але драйвера відеокарти необхідно оновити до останньої версії для покращення продуктивності.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Завданням кваліфікаційної роботи є спроектувати та розробити стратегічну відео гру «Valhalla Auto Chess» на платформі Unity, метою якої є розвинути логіку та кмітливість гравців.

Переміщення фігур відбувається в тривимірному просторі, можна рухатися вліво, вправо, вперед та назад.

На розробленій карті відображається зміст назви гри, є контролер камери, щоб гравець зміг знайти для себе оптимальне положення, для більш чіткого сприйняття інформації. Ігрова дошка розрахована для двох гравців, тобто коли відбувається хід гравця, він може взаємодіяти тільки зі своїми фігурами.

При закінчені гри відображається переможець. В будь-який момент гра може бути призупинена для редагування користувачем графіки та гучності звуків.

Головне меню містить кнопки виходу з гри та ігрових налаштувань. Додатково, для ознайомлення з правилами гри та особливостями додатку, в головному меню міститься кнопка інструкції, яка запускає записаний гемплей.

2.2. Опис застосованих математичних методів

Під час проєктування та розробки додатку жодних математичних методів не використовувалось.

2.3. Опис використаної архітектури та шаблонів проектування

Життєвий цикл програмного забезпечення - це безперервний процес, який починається з моменту прийняття рішення про необхідність створення ПЗ і закінчується в момент його повного вилучення з експлуатації.

Існує декілька підходів при визначенні фаз та робіт життєвого циклу програмного забезпечення (ЖЦПЗ), кроків процесу програмування, каскадна і спіральна моделі. Але всі вони містять загальні основні компоненти: постановка завдання, проектування рішення, реалізація, обслуговування.

В даному дипломному проекті використаю процес програмування по Райлі.

Процес програмування включає чотири кроки (рис. 2.1):

—постановка задачі, тобто отримання адекватного уявлення про те, яке завдання має виконати програма;

—проектування рішення вже поставленого завдання (загалом, таке рішення є менш формальним, ніж остаточна програма);

—кодування програми, тобто переклад спроектованого рішення в програму, яка може бути виконана на машині;

—супровід програми, тобто безперервний процес усунення в програмі неполадок і додавання нових можливостей.

Рис. 2.1. Кроки програмування по Райлі

Програмування починається з того моменту, коли користувач, тобто той, хто потребує програми для вирішення завдання, викладає проблему системного аналітику. Користувач і системний аналітик спільно визначають постановку задачі. Остання потім передається алгорітмісту, який відповідає за проектування рішення. Рішення (або алгоритм) представляє послідовність операцій, виконання яких призводить до вирішення завдання. Оскільки алгоритм часто не пристосований до виконання на машині, його слід перевести в машинну програму. Ця операція виконується кодувальником. За наступні зміни в програмі несе відповідальність супроводжуючий програміст. І системний аналітик, і алгоритміст, та кодер, і супроводжуючий програміст - всі вони є програмістами.

У разі великого програмного проекту число користувачів, системних аналітиків і алгоритмістов може виявитися значним. Крім того, може виникнути необхідність повернутися до попередніх кроків в силу непередбачених обставин. Все це служить додатковим аргументом на користь ретельного проектування програмного забезпечення: результати кожного кроку повинні бути повними, точними і зрозумілими.

Одним з найбільш важливих кроків програмування є постановка задачі. Вона виконує функції контракту між користувачем і програмістом (програмістами). Як і юридично погано складений контракт, погана постановка задачі марна. При гарній постановці завдання як користувач, так і програміст ясно і недвозначно представляють завдання, яку необхідно виконати, тобто в цьому випадку враховуються інтереси як користувача, так і програміста. Користувач може планувати використання ще нествореного програмного забезпечення, спираючись на знання того, що воно може. Гарна постановка завдання служить основою для формування її рішення.

Постановка завдання (специфікація програми), по суті, означає точне, повне і зрозуміле опис того, що відбувається при виконанні конкретної програми. Користувач зазвичай дивиться на комп'ютер, як на чорний ящик: для нього неважливо, як працює комп'ютер, а важливо, що може комп'ютер з того,

що цікавить користувача. При цьому основна увага фокусується на взаємодії людини з машиною.

Характеристики правильної постановки завдання:

1. Точність, тобто виключення будь-якої неоднозначності. Не повинно виникати питань щодо того, яким буде висновок програми при кожному конкретному введенні.

2. Повнота, тобто розгляд всіх варіантів для заданого введення, включаючи помилковий або непередбачений введення, і визначення відповідного висновку.

3. Ясність, тобто вона повинна бути зрозумілою і користувачеві і системного аналітику, оскільки постановка завдання - це єдиний контракт між ними.

Д. Райлі пропонує для постановки задачі користуватися стандартною формою, яка забезпечує максимальну точність, повноту, ясність і включає:

— найменування завдання (схематичне визначення);

— загальний опис (короткий виклад завдання);

— введення;

— висновок;

— помилки (явно перераховані незвичайні варіанти введення, щоб показати користувачам і програмістам ті дії, які зробить машина в подібних ситуаціях);

— приклад (хороший приклад може передати сутність завдання, а також проілюструвати різні випадки).

Проектування рішення програмування є найбільш важким. На даній стадії постановка задачі має бути перетворена на алгоритм. Тому алгорітміст повинен володіти достатнім досвідом програмування і підходити до кожної нової задачі, спираючись на твердо встановлену методику проектування. Щоб уникнути помилок в програмах, алгорітмісти повинні використовувати ретельно розроблені процедури конструювання, засновані на правилах логічного висновку.

Завдання проектувальника - створення алгоритму, що виконує функції сполучної ланки між постановкою завдання і готова для виконання програмою. Перевірку створеного алгоритму, тобто наскільки останній відображає постановку завдання, здійснює системний аналітик. У силу цього і системний аналітик, і проектувальник повинні вміти читати і розуміти алгоритм. Кожен алгоритм записується на деякій псевдомова. Алгоритми, звані також псевдокод, не можуть бути виконані ні на якому комп'ютері.

Кодування алгоритму полягає в перекладі алгоритму в програму. Для створення повної, точної та зрозумілої програми необхідні відповідні методи запису програм. На відміну від природних, мови програмування створені спеціально для такого подання рішення завдання, яке може бути виконано комп'ютером.

Перш ніж завершити роботу, кодіровщік повинен переконатися, що програма відповідає псевдокод. Потім системний аналітик, алгорітміст і, що найголовніше, користувач повинні протестувати і підтвердити, що вона працює правильно. Після цього можна вважати, що програма готова для передачі користувачеві в комплекті з усією необхідною документацією.

Однак на цьому програмування не закінчується, далі слідує крок супроводу. Справа в тому, що в програмі можуть бути помилки, зумовлені або неадекватною постановкою завдання, або тим, що проект не задовольняє постановці задачі або програма не відповідає проекту. Яка б не була причина, користувач має право вимагати коригування програми, оскільки він не уявляв, що програма буде працювати таким чином. Виправлення помилок є однією з головних завдань супроводу програм. Інший не менш важливим завданням супроводу програм є її модифікація, тобто додавання в програму нових можливостей або зміна існуючих. Користувач може змінити вимоги до роботи програми, що, у свою чергу, призведе до необхідності її переписати. Складність операцій із супроводу програми залежить від типу змін, які повинні бути зроблені: у гіршому випадку може знадобитися повна переробка програми від постановки до кодування. Зазвичай на супровід програми витрачається більше

часу, ніж на її створення.

Останньою складовою процесу програмування є документування. Воно включає широкий спектр описів, що полегшують процес програмування і збагачують результуючу програму. Постійне документування має становити невід'ємну частину кожного кроку програмування. Постановка завдання, проектні документи, алгоритми і програми - все це документи. Внутрішня документація, зазначена безпосередньо в програму, полегшує читання коду.

Відповідно до моделі, представленої в цьому проекті, програмування можна розділити на чотири кроки: постановку задачі, проектування рішень, кодування програми, супровід програми. Додатково модель включає документування програми як дії, які необхідно виконувати протягом всього процесу програмування.

2.4. Опис використаних технологій та мов програмування

2.4.1. Опис використаних засобів програмування

Відеогра повинна повноцінно функціонувати в операційній системі Windows.

Основні характеристики Windows:

– графічний інтерфейс: всі елементи, які користувач бачить на екрані (вікна, кнопки, смуги скролінгу), знаходяться в файлах ОС. Існує безліч бібліотек, які містять функції управління цими елементами, що дозволяє програмісту не відволікатися на створення інтерфейсу програми і стандартизує зовнішній вигляд додатків;

– багатозадачність: Windows підтримує одночасне виконання декількох програм на рівнях процесів (програм) і потоків (всередині програмний паралелізм), забезпечення яких відбувається за участю апаратного рівня. ОС містить велику кількість функцій, що забезпечують управління паралельно виконуваними програмами, синхронізацію, поділ адресного простору пам'яті між додатками тощо;

– апаратно-незалежне програмування: використання функцій операційної системи в прикладній програмі дозволяє програмісту не піклуватися про сумісність програми з апаратурою. ОС сама (за допомогою драйверів) виконує узгодження введення-виведення. Таким чином, програма, створена на одній конфігурації буде працювати на інший;

– події і механізм повідомлень: в більшості випадків програма починає працювати тільки при спрацьовуванні деякої події в системі (запуск програми, необхідність перемальовування вікна програми, сигнал від таймера тощо). В інший час програма або взагалі не виконується, або виконує невеликий код в фоновому режимі. Більш того, ОС не прямо викликає деяку процедуру програми для відпрацювання реакції на якусь подію, а посилає відповідне повідомлення в програму, і вона сама вирішує, що їй робити;

– віконне середовище: частіше всього в кожній програмі є одне або декілька вікон, які мають стандартний вигляд і містять обов'язкові елементи (наприклад, іконку, рядок заголовка, схоже меню, кнопки закриття, мінімізації та максимізації).

Для запуску гри необхідна версія ОС Windows не нижче 7 SP1, але для кращої стабільності необхідно використовувати більш сучасні версії.

Unity3d є сучасним крос-платформним движком для створення ігор і додатків, розроблений Unity Technologies. За допомогою даного движка можна розробляти не тільки додатки для комп'ютерів, але і для мобільних пристроїв, ігрових приставок і інших девайсів. Інтерфейс платформи зображено на рис. 2.2.

Рис. 2.2. Інтерфейс платформи Unity

Характеристики платформи:

– по-перше, в середовище розробки Unity інтегрований ігровий движок, за допомогою якого можна протестувати свою гру не виходячи з редактора;

– по-друге, Unity підтримує імпорт величезної кількості різних форматів, що дозволяє розробнику гри конструювати самі моделі в більш зручному додатку, а Unity використовувати за прямим призначенням - розробки продукту;

– по-третє, написання сценаріїв (скриптів) здійснюється на найбільш популярних мовах програмування - C# і JavaScript [1].

Таким чином, Unity3d є актуальною платформою, за допомогою якої можна створювати свої власні додатки і експортувати їх на різні пристрої, будь то мобільний телефон або приставка Nintendo Wii.

Для того щоб створити свою гру, потрібно володіти однією з доступних (на Unity) мов програмування: C #, JavaScript або Boo.

Їх компоненти мають ряд властивостей або змінних, які можна налаштувати як у вікні Inspector редактора Unity, так і за допомогою скрипта.

Unity надає багато влаштованих компонентів, але можна створити власний для реалізації специфічних алгоритмів. Це можна зробити за допомогою створення користувацького скрипта та прикріплення його до необхідного об’єкта. Кожен скрипт зв'язується з внутрішніми механізмами Unity шляхом реалізації класу, похідного від вбудованого класу MonoBehaviour.

Компоненти на основі скриптів дозволяють запускати ігрові події, перевіряти об'єкт на предмет колізій, застосовувати фізичні властивості, програмувати реакцію на управління користувача і багато іншого.

C# є об'єктно-орієнтованою мовою, але підтримує також і компонентно-орієнтоване програмування. Розробка сучасних додатків все більше тяжіє до створення програмних компонентів у формі автономних і самоописувальних пакетів, що реалізують окремі функціональні можливості. Головна особливість таких компонентів в тому, що вони являють собою модель програмування з властивостями, методами і подіями. У них є атрибути, що надають декларативні відомості про компоненті. Вони включають в себе власну документацію. C # надає мовні конструкції, безпосередньо підтримують, таку концепцію роботи. Завдяки цьому C # підходить для створення і застосування програмних компонентів.

Ось лише кілька функцій мови C #, що забезпечують надійність і стійкість додатків. Прибирання сміття автоматично звільняє пам'ять, зайняту недосяжними невикористовуваними об'єктами. Обробка винятків надає структурований і розширюваний підхід до виявлення помилок і їх відновлення. Типобезпечна структура мови унеможливлює читання з не ініціалізованих змінних, індексацію масивів за межами їх кордонів або виконання неперевірених привидів типів.

У C# всі типи даних, включаючи типи-примітиви, такі як int і double, успадковують від одного кореневого типу object. Таким чином, всі типи використовують загальний набір операцій і значення будь якого типу можна зберігати, передавати і обробляти схожим чином. Крім того, C# підтримує призначені для користувача посилальні типи і типи значень, дозволяючи як

динамічно виділяти пам'ять для об'єктів, так і зберігати спрощені структури в стек.

Мова програмування забороняє звернення до змінних, що не були ініційовані, що виключає можливість виконання безконтрольного приведення типів або виходу за межі певного масиву даних.

Для роботи додатків на C# необхідно встановити і налаштувати платформу NET Framework. Платформа вбудована в інсталяційний пакет Windows, при необхідності її також можна скачати та інсталювати окремо. Існують також версії для Linux і MAC.

В рамках платформи до обробки виконуваного коду під'єднується середовище CLR – єдиний об'єднаний набір бібліотек і класів, який був розроблений Майкрософт і є реалізацією світового стандарту

Common Language Infrastructure (CLI).

Основний механізм CLR має вигляд бібліотеки під назвою mscoree.dll (називається загальним механізмом виконання виконуваного коду об'єктів – Common Object Runtime Execution Engine). При додаванні посилання на збірку для її використання, завантаження бібліотеки mscoree.dll здійснюється автоматично і потім, в свою чергу, призводить до завантаження необхідної збірки в пам'ять.

Щоб забезпечити сумісність програм і бібліотек C # при подальшому розвитку, при розробці C # багато уваги було приділено управлінню версіями. Багато мови програмування обходять увагою це питання. В результаті програми на цих мовах ламаються частіше, ніж хотілося б, при виході нових версій залежних бібліотек. Питання управління версіями істотно вплинули на такі аспекти розробки C#, як роздільні модифікатори virtual і override, правила вирішення перевантаження методів і підтримка явного оголошення членів інтерфейсу.

У недавніх версіях C# були використані інші парадигми програмування. C# включає функції, що підтримують прийоми функціонального програмування, такі як лямбда-вирази. Інші нові можливості підтримують поділ

даних і алгоритмів, наприклад зіставлення шаблонів.

Організаційна структура C# ґрунтується на таких поняттях, як: - типи і змінні;

– класи є найважливішим типом в мові C#. Об'єкти представляють собою екземпляри класів. Класи створюються описом їх членів;

– масив - це структура даних, що містить кілька змінних, доступ до яких здійснюється за який обчислюється індексам;

– інтерфейс визначає контракт, який може бути реалізований класами і структурами. Інтерфейс може містити методи, властивості, події і індексатори. Інтерфейс не надає реалізацію членів, які в ньому визначені. Він лише перераховує члени, які повинні бути визначені в класах або структурах, що реалізують цей інтерфейс;

– delegate представляє посилання на методи з конкретним списком параметрів і типом значення, що повертається. Делегати дозволяють використовувати методи як сутності, зберігаючи їх у змінні і передаючи в якості параметрів. Принцип роботи делегатів близький до покажчиків функцій з деяких мов, але на відміну від покажчиків функцій делегати є об'єктно-орієнтованими і строго типізовані;

– атрибути дозволяють програмам вказувати додаткові описові да-ні про типи, членах та інших сутності [2].

Інтегроване середовище розробки Visual Studio - це стартовий майданчик для написання, налагодження і складання коду, а також подальшої публікації додатків. Інтегроване середовище розробки (IDE) являє собою багатофункціональну програму, яку можна використовувати для різних аспектів розробки програмного забезпечення. Крім стандартного редактора і відладчика, які існують в більшості середовищ IDE, Visual Studio включає в себе компілятори, засоби авто-завершення коду, графічні конструктори і багато інших функцій для спрощення процесу розробки. Інтерфейс середи (рис.2.3).

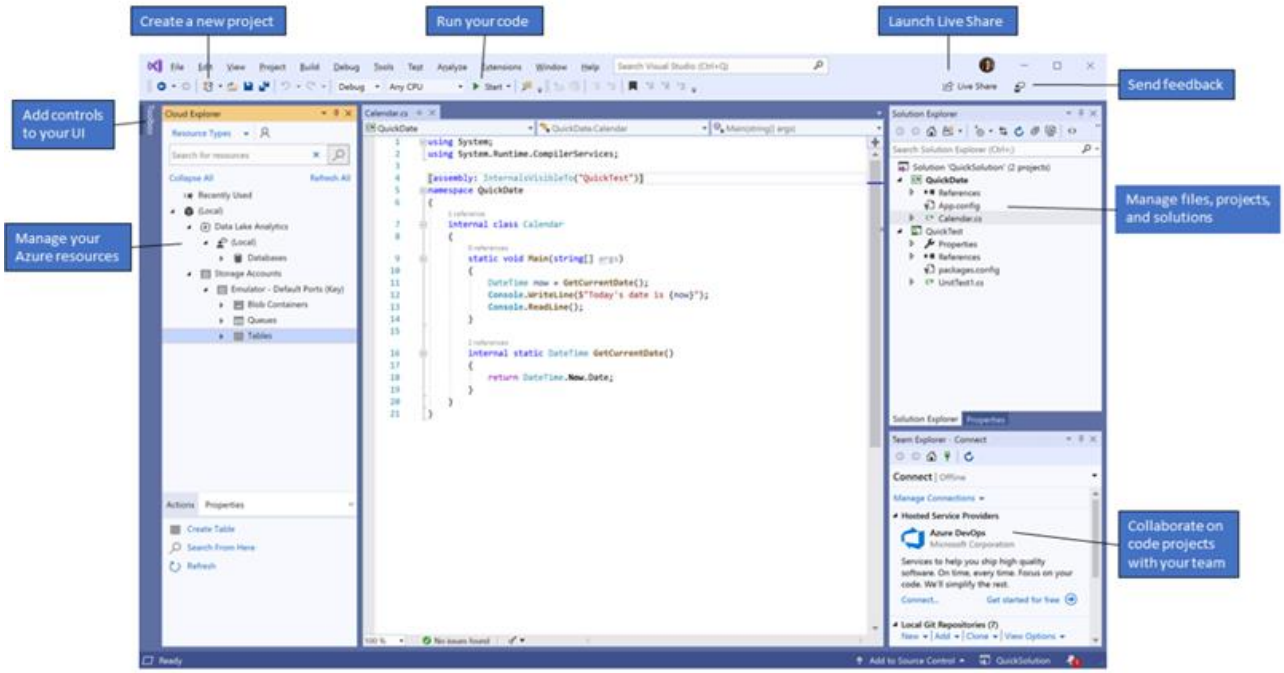

Рис. 2.3. Інтерфейс середовище Visual Studio

На рисунку показано середовище Visual Studio з відкритим проектом і декількома вікнами основних інструментів.

Оглядач рішень (вгорі праворуч) дозволяє переглядати файли коду, пересуватися по ньому і управляти ними. Оглядач рішень дозволяє впорядкувати код шляхом об'єднання файлів в рішення і проекти.

У вікні редактора (центр), відображається вміст файлу. Тут можна редагувати код або розробляти призначений для користувача інтерфейс, наприклад вікно з кнопками або текстові поля.

Team Explorer (правий нижній кут) дозволяє відслідковувати робочі елементи і використовувати код спільно з іншими користувачами за допомогою технологій управління версіями, таких як Git і система управління версіями Team Foundation (TFVC).

Функціональність Visual Studio охоплює всі етапи розробки програмного забезпечення, надаючи сучасні інструменти для написання коду, проектування графічних інтерфейсів, збірки, налагодження і тестування програм. Можливості Visual Studio можуть бути доповнені шляхом підключення необхідних

розширень.

Хвилясті лінії позначають помилки або потенційні проблеми коду прямо під час введення. Ці візуальні підказки дозволяють усувати проблеми негайно і не чекати, поки помилка буде виявлена під час збирання або запуску програми. Якщо навести курсор миші на хвилясту лінію, на екран будуть виведені додаткові відомості про помилку. Крім того, в полі зліва може з'являтися значок лампочки з швидкими діями щодо усунення помилки.

Можна одним натисканням кнопки відформатувати код і застосувати до нього виправлення, запропоновані параметрами стилю коду, які вказані в файлі EditorConfig. Очищення коду допомагає усунути багато проблем в коді ще до перевірки.

Рефакторинг включає в себе такі операції, як інтелектуальне перейменування змінних, витягування однієї або декількох рядків коду в новий метод, зміна порядку параметрів методів і багато іншого.

Редактор коду Visual Studio підтримує підсвічування синтаксису, вставку фрагментів коду, відображення структури і пов'язаних функцій. Істотно прискорити роботу допомагає технологія IntelliSense – автозавершення коду під час введення.

Вбудований відладчик Visual Studio використовується для пошуку і виправлення помилок у вихідному коді, в тому числі на низькому апаратному рівні. Інструменти діагностики дозволяють оцінити якість коду з точки зору продуктивності і використання пам'яті.

Visual Studio надає комплекс інструментів для автоматизації тестування додатків в частині перевірки роботи інтерфейсів, модульного і навантажувального тестування.

Система налагодження Visual Studio дозволяє переглядати код з кроком в одну інструкцію, перевіряючи значення змінних. Можна задати точки зупину, які зупиняють виконання коду на певному рядку. Таким чином можна побачити як значення змінної змінюється по мірі виконання коду [3].

Blender – це безкоштовне програмне забезпечення для створення і

редагування тривимірної графіки.

В роботі використовується для створення моделей оточення гри, які в подальшому будуть розміщені на ігрових рівнях за допомогою інструментів проектування платформи Unity.

Підтримка різноманітних геометричних примітивів, включаючи полігональні моделі, систему швидкого моделювання в режимі subdivision surface (SubSurf), криві Безьє, поверхні NURBS, metaballs (метасфери), скульптурне моделювання та векторні шрифти.

Універсальні вбудовані механізми рендеринга і інтеграція з зовнішніми рендерер YafRay, LuxRender і багатьма іншими.

Інструменти анімації, серед яких інверсна кінематика, скелетна анімація і сіткова деформація, ключові кадри, нелінійна анімація, редагування вагових коефіцієнтів вершин, обмежувачі.

Blender Game Engine - підпроект Blender, що надає інтерактивні функції, такі як визначення колізій, движок динаміки і програмована логіка. Також він дозволяє створювати окремі real-time-додатки починаючи від архітектурної візуалізації до відео ігор. Інтерфейс програми зображено на рис. 2.4.

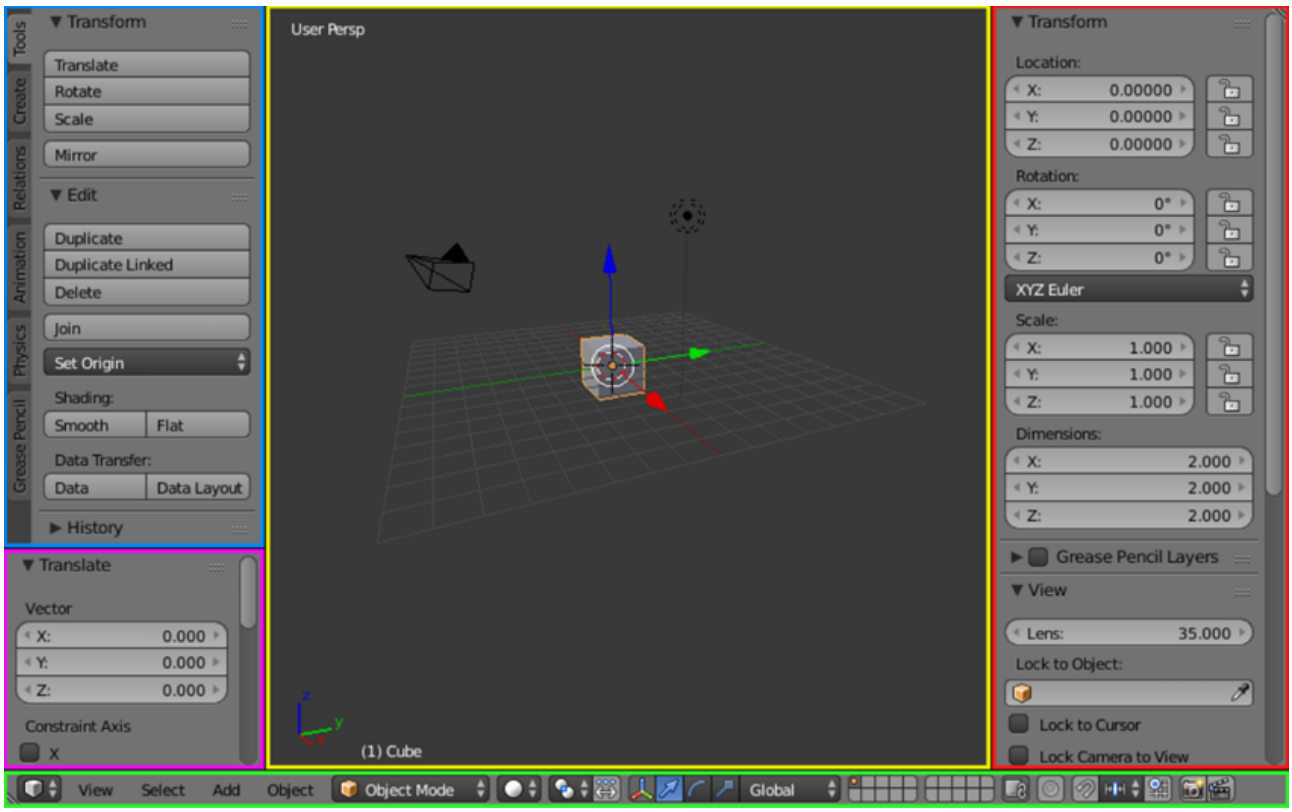

Рис. 2.4. Вікно програми Blender

2.4.2. Аналіз алгоритмів рішення логічних ігор

2.4.2.1. Основні методи побудови алгоритмів

Метод «грубої сили» (від англ. Brute force; або повний перебір) - метод рішення задачі шляхом перебору всіх можливих варіантів. Складність повного перебору залежить від кількості всіх можливих рішень задачі. Якщо простір рішень дуже великий, то повний перебір може не дати результатів протягом декількох років або навіть століть.

Будь-яка задача з класу NP може бути вирішена повним перебором. При цьому, навіть якщо обчислення цільової функції від кожного конкретного можливого рішення задачі може бути здійснене за поліноміальний час, в залежності від кількості всіх можливих рішень повний перебір може зажадати експоненціального часу роботи.

На методі грубої сили базується атака повного перебору - вид криптоаналізу, який полягає у переборі ключів, з множини можливих.

Методи оптимізації повного перебору:

1. Метод гілок і меж. Для прискорення перебору метод гілок і меж використовує відсів підмножин допустимих рішень, про які наперед відомо що вони не містять оптимальних рішень.

2. Розпаралелювання обчислень. Для збільшення швидкості підбору ключа використовується розпаралелювання обчислень. Існує два підходи до розпаралелювання:

- побудова конвеєра. Нехай алгоритм співвідношення можна уявити у вигляді ланцюжка найпростіших дій (операцій). Візьмемо N процесорів, задамо їх порядок, і-ий процесор виконує три однакові за часом операції:

а) отримання даних від - і-1-го процесора;

б) виконання операції;

в) передача даних і+1-му процесору.

Тоді конвеєр з N послідовно з'єднаних, паралельно і синхронно працюючих процесорів працює зі швидкістю v/3, де v - швидкість виконання однієї операції одним процесором;

- другий підхід полягає що множина K всіх можливих ключів розбивається на непересічні підмножини. Система з Q машин перебирає ключі так, що і-та машина здійснює перебір ключів з множини. Система припиняє роботу, якщо одна з машин знайшла ключ. Найважче - це розділення вихідної множини. Але якщо кожен процесор почне обчислення з якогось довільного ключа, то час перебору збільшиться, але схема значно спроститься. Середнє число кроків у цьому випадку становить |K|/N, де |K| - число елементів у множині ключів, а N - число процесорів.

Самий простий алгоритм рішення задачі - це метод повного перебору.

Стратегія пошуку найкращого ходу здійснюється за допомогою рекурсії. Рекурсія – це коли функція визиває сама себе, й кожен раз при вході в функцію у стек програми записується копія змінних. Але не відноситься до статичних змінних, спільних для усіх викликів функції.

Рекурсія дозволяє описати деякі процеси, лише визначивши закони їх розвитку, а як в цілому буде розвиватись алгоритм, не так важливо. Деякі завдання, які легко вирішуються за допомогою рекурсії, часто неможливо або складно вирішити іншим способом. Теж саме відноситься і до пошуку здійснення найкращого ходу. Для цього потрібно отримати всі варіанти ходів (переміщення), зробити оцінку для кожного з них та знайти переміщення з максимальною оцінкою.

Основний алгоритм оптимізації перебору є так званий метод alpha-beta. Сенс його полягає в тому, що для отримання оцінки тієї самої точності, як і при повному переборі, зовсім необов'язково розглядати всі варіанти.

Альфа-бета алгоритм, при найкращому порядку ходів, побудує значно менше дерево перебору. Це приблизно дорівнює кореню квадратному з числа позицій, що переглядаються при повному переборі. Альфа-бета дуже чутливий до порядку ходів. Тому потрібно врахувати, що при найгіршому порядку ходів, тобто коли відсічення за beta викликає останній хід, альфа-бета прогляне стільки ж позицій, що і мінімакс. Швидкість прорахунку також дуже залежить на практиці від можливого діапазону оцінок. Наприклад, коли враховується тільки матеріал, то оцінка всім ходам, крім узяття, буде дорівнювати нулю. Це означає, що відсічень буде дуже багато, особливо якщо взяття розглядатимуться першими.

2.4.2.2. Алгоритм «Мінімакс»

Мінімакс – правило прийняття рішень, що використовується в теорії ігор для мінімізації можливих втрат з тих, які особа, яка приймає рішення не може уникнути при розвитку подій за найгіршим для неї сценарієм.

У комбінаторній теорії ігор є алгоритм мінімаксу для ігрових рішень. Простий варіант алгоритму мінімаксу, зазначений нижче, стосується ігор, таких як гра в хрестики-нулики, де кожен гравець може виграти, програти або зіграти в нічию. Якщо гравець А може виграти за один хід, то це і є його найкращий

хід. Якщо гравець Б знає, що певний крок призведе до ситуації, коли гравець А може виграти в один хід, в той час інший крок призведе до ситуації, коли гравець А може, за найсприятливіших обставин, претендувати на нічию, то найкращий хід гравця Б є один з тих, що веде до нічиї. Наприкінці гри легко побачити, які кроки є «найкращими». Алгоритм мінімаксу допомагає знайти найкращий хід, працюючи в зворотному порядку від кінця гри. На кожному кроці він припускає, що гравець А намагається максимізувати шанси на перемогу, а на наступний хід гравець Б намагається звести ці шанси до мінімуму (тобто максимізувати власні шанси на перемогу).

У теорії ігор максимін часто відрізняється від мінімаксу. Мінімакс використовується в іграх з нульовою сумою, щоб підкреслити мінімізацію максимального виграшу противника. У грі з нульовою сумою, це ідентично максимізації свого виграшу. «Максимін» - це термін, який зазвичай використовується для ігор з ненульовою сумою, щоб описати стратегію, яка максимізує власну мінімальну винагороду. У іграх з ненульовою сумою це не те ж саме, що мінімізація максимальної винагороди противника, і не стратегія рівноваги Неша.

У комбінаторній теорії ігор є алгоритм мінімаксу для ігрових рішень. Простий варіант алгоритму мінімаксу, зазначений нижче, стосується ігор, таких як гра в хрестики-нулики, де кожен гравець може виграти, програти або зіграти в нічию. Якщо гравець А може виграти за один хід, то це і є його найкращий хід. Якщо гравець Б знає, що певний крок призведе до ситуації, коли гравець А може виграти в один хід, в той час інший крок призведе до ситуації, коли гравець А може, за найсприятливіших обставин, претендувати на нічию, то найкращий хід гравця Б є один з тих, що веде до нічиї. Наприкінці гри легко побачити, які кроки є «найкращими». Алгоритм мінімаксу допомагає знайти найкращий хід, працюючи в зворотному порядку від кінця гри. На кожному кроці він припускає, що гравець А намагається максимізувати шанси на перемогу, а на наступний хід гравець Б намагається звести ці шанси до мінімуму (тобто максимізувати власні шанси гравця Б на перемогу).

Алгоритм мінімаксу з альтернативним рухом: мінімаксний алгоритм рекурсивний алгоритм для вибору наступного кроку для ігор з n гравцями, як правило, двох гравців. Евристична оцінка пов'язана з кожним положенням або станом гри. Це значення обчислюється за допомогою евристичної функції, і це показує, як добре було б для гравця досягти цієї позиції. Потім гравець робить хід, який максимізує мінімальне значення позиції в результаті можливих наступних ходів суперника. Якщо черга А робити хід, то А дає оцінку для кожного зі своїх ходів.

Можливий спосіб розрізнення полягає в призначенні для певної гри характеристик для А як +1 і для Б як -1. Альтернативою є використання правила, що якщо в результаті руху А є безпосередня перемога А, то призначити певну додатну оцінку, а якщо призведе до безпосередньої перемоги Б , певну від'ємну оцінку. Значення А для будь-якого іншого ходу є мінімум значень, отриманих від кожного з можливих ходів Б. З цієї причини, А називають максимізуючим гравцем і Б називають мінімізуючим гравцем, звідси і назва алгоритму мінімаксу. Алгоритм призначить певне додатне чи від'ємне значення для будь-якого стану, оскільки значення кожної позиції буде знаходитись із значень деяких остаточних виграшних чи програшних позицій. Часто це стає можливим тільки в самому кінці складних ігор, таких як шахи або го, так як це не обчислювально дивитися в майбутнє аж до завершення гри, за винятком ситуацій, близьких до кінця гри.

Але це може бути знайдено, якщо ми зможемо поставити евристичну функцію оцінки, яка дає значення для некінцевих станів гри без урахування всіх можливих наступних станів. Тоді ми можемо ввести обмеження для мінімаксного алгоритму - дивитися тільки на певну кількість ходів вперед. Це число називається "глибиною пошуку".

Використання алгоритмів евристичного пошуку для пошуку на графі виграшної стратегії в складніших задачах та іграх (шашки, шахи) не реальний. За деякими оцінками ігрове дерево гри в шашки містить 1040 вершин, в шахах 10120 вершин. Якщо при грі в шашки для однієї вершини потрібно 1/3

наносекунди, то всього ігрового часу буде потрібно 1021 століть. У таких випадках вводяться штучні умови зупинки, засновані на таких факторах, як найбільша допустима глибина вершин у дереві пошуку або обмеження на час і обсяг пам'яті. Наприклад, шаховий комп'ютер Deep Blue (який виграв у Гарі Каспарова) для глибини у 12 ходів перебравши усі можливі стани, тоді застосував евристичні функції оцінки.

Алгоритм може розглядатися як дослідження вузла дерева варіантів. Фактором розгалуження дерева є середнє число дітей для кожного вузла (тобто середня кількість можливих ходів для будь-якого стану). Число вузлів зазвичай зростає експотенційно з глибиною пошуку. Число вузлів, які аналізуються, є приблизний коефіцієнт розгалуження зведений в ступінь глибини пошуку. Тому непрактично повністю проаналізовувати такі ігри, як шахи, використанням мінімаксного алгоритму.

Продуктивність простого алгоритму мінімаксу може бути значно покращено, не впливаючи на результат, за допомогою відсічення альфа-бета. Теоретично, це еквівалент алгоритму мінімаксу, за допомогою якого завжди виходить такий же результат, але помітно швидше, так як цілі частини дерева виключаються без проведення аналізу. В основі цієї процедури лежить ідея Дж. Маккарті про використання двох змінних, позначених α і β (1961 рік).

Інші методи евристичних відсікань також можуть бути використані, але не всі з них гарантовано дають той же результат, що й алгоритм без відсікань.

Простий алгоритм мінімаксу може бути тривіально змінений, щоб додатково повертати саму стратегію разом з результатом мінімаксу.

2.4.2. 3. Застосування теорії Шпраг-Гранді

Перейдемо тепер до найголовнішого в даній роботі - теоремі про еквівалентність німу будь-якої рівноправної гри двох гравців.

Теорема Шпраг-Гранді. Розглянемо будь-який стан v деякої рівноправної гри двох гравців. Нехай з нього є переходи до деяких стану $v_i (i = 1..k), k \geq 0$. Стверджується, що станом v цієї гри можна поставити у відповідність купку нима деякого розміру x (яка буде повністю описувати стан v нашої гри - тобто ці два стани двох різних ігор будуть еквівалентні). Це число x - називається значенням Шпраг-Гранді стану v. Більш того, це число x можна знаходити наступним рекурсивним чином: порахуємо значення Шпраг-Гранді x_i що кожному переходу (v, v_i), і тоді виконується:

$$x = mex\{x_1, ..., x_k\}, \tag{2.1}$$

де функція mex від безлічі чисел повертає найменше невід'ємне число, що не зустрічається в цій множині (назва "mex" - це скорочення від "minimum excludant").

Таким чином, ми можемо, стартуючи від вершин без вихідних ребер, поступово порахувати значення Шпраг-Гранді для всіх станів нашої гри. Якщо значення Шпраг-Гранді будь-якого стану дорівнює нулю, то це стан програшно, інакше - виграшно.

Доведення. Для вершин, з яких немає жодного переходу, величина x відповідно до теореми буде виходити як mex від порожнього безлічі, тобто x = 0. Але, справді, стан без переходів - це програшна стан, і йому дійсно повинна відповідати ним-купка розміру 0. Розглянемо тепер будь-який стан v, з якого є переходи. За індукції ми можемо вважати, що для всіх станівVi, в які ми можемо перейти з поточного стану, значення Xi вже підраховані.

Порахуємо величину $p = mex\{x_1, ..., x_k\}$. Тоді, згідно з визначенням функції mex, ми отримуємо, що для будь-якого числа i в проміжку [0; p) знайдеться хоча б один відповідний перехід в якесь із Vi-их стан. Крім того, можуть існувати також додаткові переходи - в стану зі значеннями Гранді, великими p. Це означає, що поточний стан еквівалентно станом німу зі збільшенням з купкою розміру p: справді, у нас є переходи з поточного стану в стану з

купками всіх менших розмірів, а також можуть бути переходи в стани великих розмірів.

Отже, величина $mex\{x_1,...,x_k\}$ дійсно є шуканим значенням Шпраг-Гранді для поточного стану, що й треба було довести.

Застосування теореми Шпраг-Гранді: наведемо нарешті цілісний алгоритм, який можна застосовувати до будь-якої рівноправній грі двох гравців для визначення виграшності / програшну поточного стану v. Функція, яка кожномустаном гри ставить у відповідність ним-число, називається функцією Шпраг-Гранді.

Отже, щоб порахувати функцію Шпраг-Гранді для поточного стану гри, потрібно:

1. Виписати всі можливі переходи з поточного стану.
2. Кожен такий перехід може вести або в одну гру, або в суму незалежних ігор.
3. У першому випадку - просто порахуємо функцію Гранді рекурсивно для цього нового стану.
4. У другому випадку, коли перехід з поточного стану надає суму декількох незалежних ігор - рекурсивно порахуємо для кожної з цих ігор функцію Гранді, а потім скажемо, що функція Гранді суми ігор дорівнює XOR-сумі значень цих ігор.
5. Після того, як ми порахували функцію Гранді для кожного можливого переходу - вважаємо mex від цих значень, і знайдене число - і є шукане значення Гранді для поточного стану.
6. Якщо отримане значення Гранді дорівнює нулю, то поточний стан програшно, інакше - виграшно.

Таким чином, у порівнянні з теоремою Шпраг-Гранді тут ми враховуємо те, що в грі можуть бути переходи з окремих станів в суми кількох ігор. Щоб працювати з сумами ігор, ми спочатку замінюємо кожну гру її значенням Гранді, тобто однією ним-купкою деякого розміру. Після цього ми приходимо

до суми декількох ним-купок, тобто до звичайного німу, відповідь для якого, відповідно до теореми Бутона - XOR-сума розмірів купок.

Дуже часто при вирішенні конкретних завдань, коли потрібно навчитися рахувати функцію Шпраг-Гранді для заданої гри, допомагає вивчення таблиць значень цієї функції в пошуках закономірностей. У багатьох іграх, які здаються важкими для теоретичного аналізу, функція Шпраг-Гранді на практиці виявляється періодичної, або ж має дуже простий вигляд, який легко помітити "на око". У переважній більшості випадків побачені закономірності є вірними, і при бажанні доводить за допомогою математичної індукції.

2.5. Опис структури програми та алгоритмів її функціонування

2.5.1. Загальний опис структури програми

Гра розрахована для двох гравців. Один гравець грає за добру сторону, а інший за ворожу. Для кожного гравця виділена своя половина ігрової дошки та свої фігури. Основна мета гри полягає в тому, щоб захистити свого короля та вбити ворожого.

Взаємодію користувачів з системою графічно відображено на діаграмі використання (рис.2.5 та 2.6).

Рис. 2.5. Діаграма використання

Рис. 2.6. Діаграма діяльності

Склад файлової системи розробленого проєкту зображено на рис. 2.7:

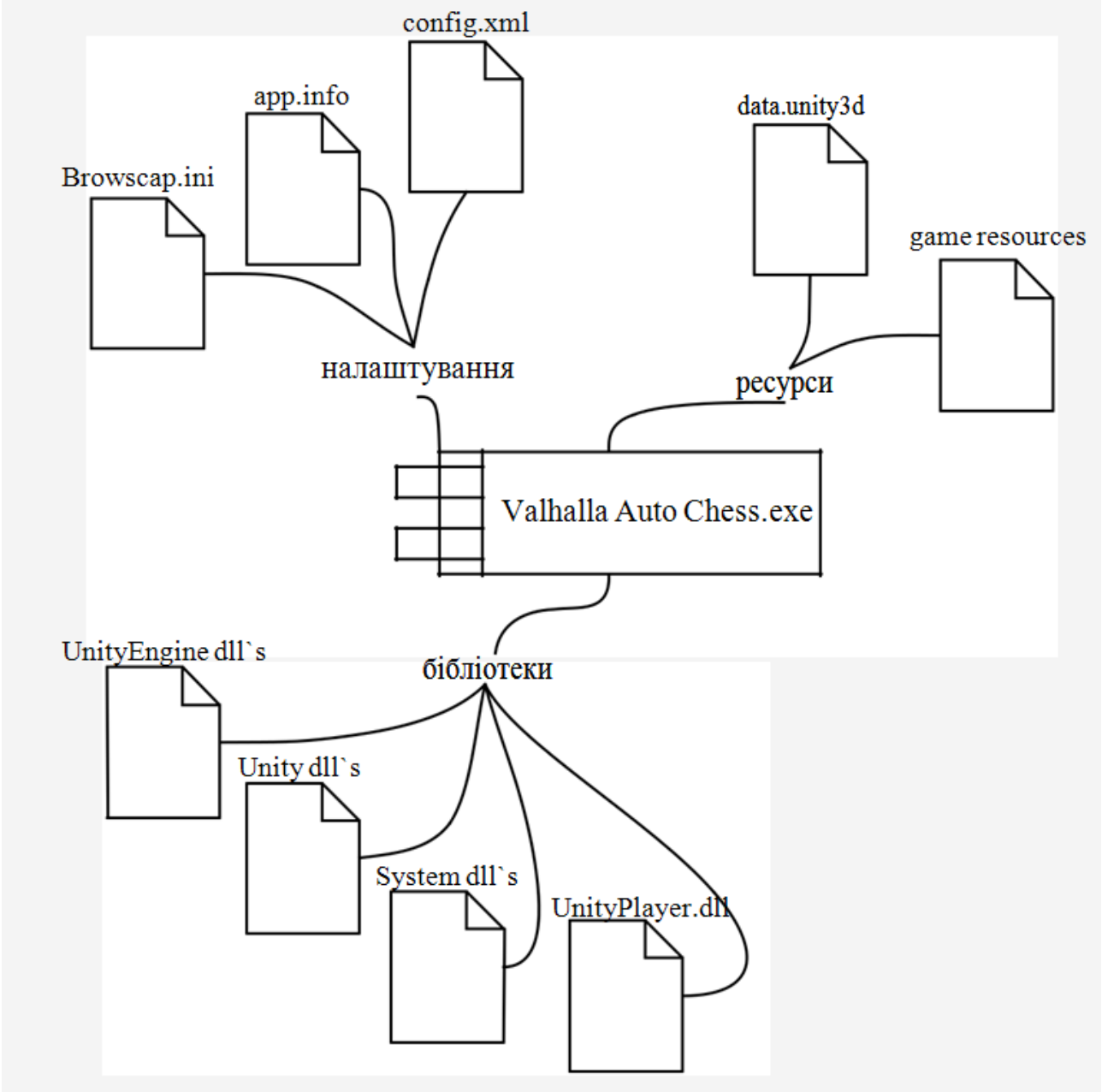

Рис. 2.7. Файлова система проєкту

Діаграми класів проєкту наведено на рис. 2.8 та 2.9.

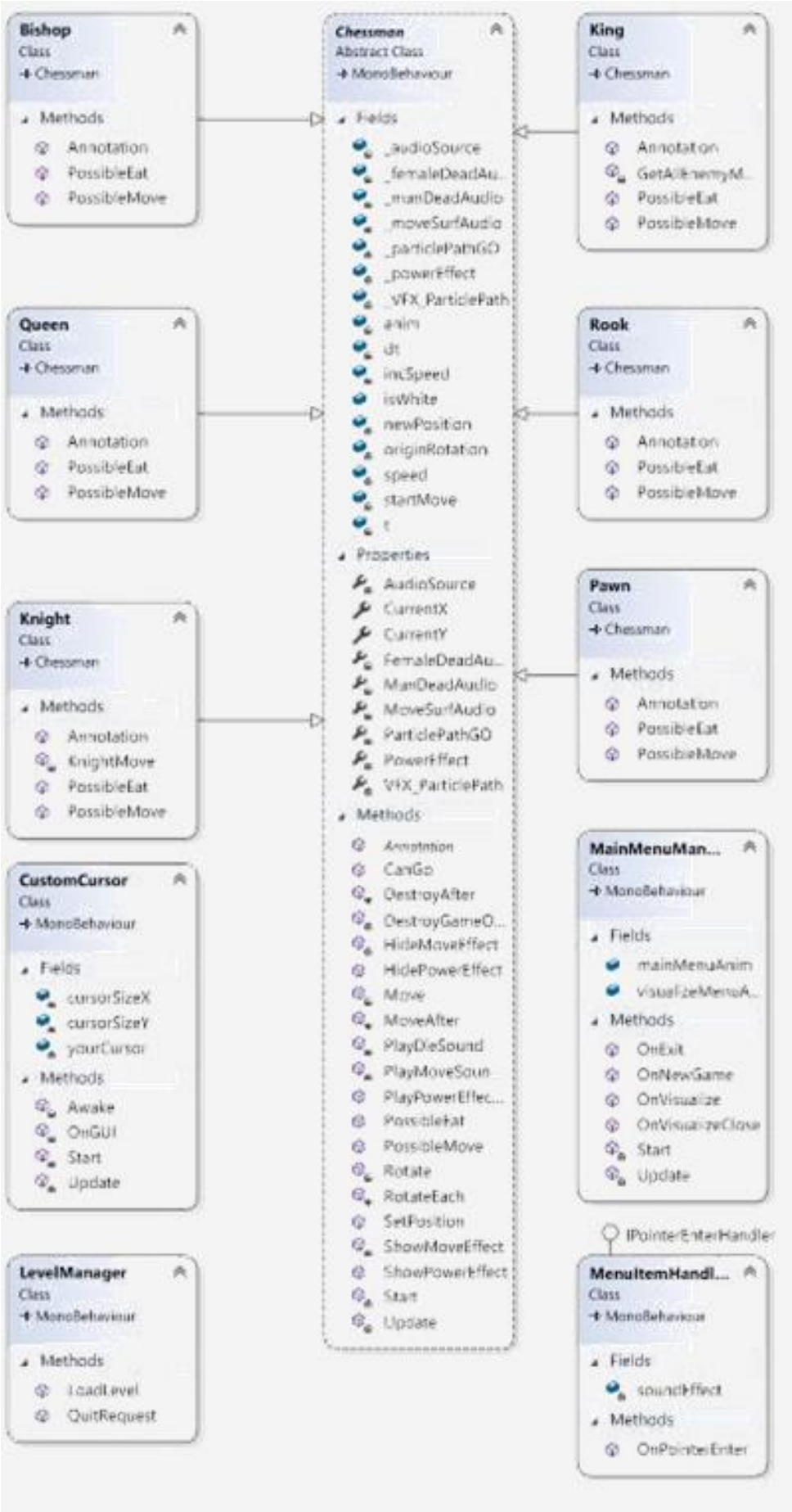

Рис. 2.9. Діаграма класів

Рис. 2.9. Діаграма класів (продовження)

2.5.2 Опис основних компонентів

В платформі Unity кожен об’єкт представляє собою GameObject, який зберігає атрибути користувацьких та влаштованих компонентів. Так, наприклад, стандартним компонентом GameObject, який не можливо видалити є «Transform» (рис. 2.10). Він зберігає параметри Position, Rotation, та Scale відповідно позиція, поворот і розмір за трьома осями (X,Y,Z) відносно початку координат.

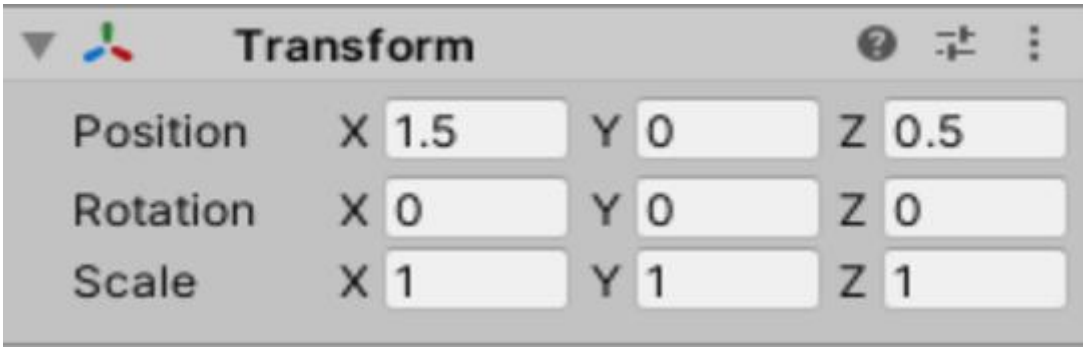

Рис. 2.10. Компонент «Transform»

Щоб відобразити двомірні зображення в тривимірному ігровому просторі використовується компонент «Sprite renderer» (рис. 2.11) Після вказівки у властивості Sprite посилання на імпортоване в проект гри над зображення можна проводити різні маніпуляції, такі як обертання, переміщення, зміна кольору тощо.

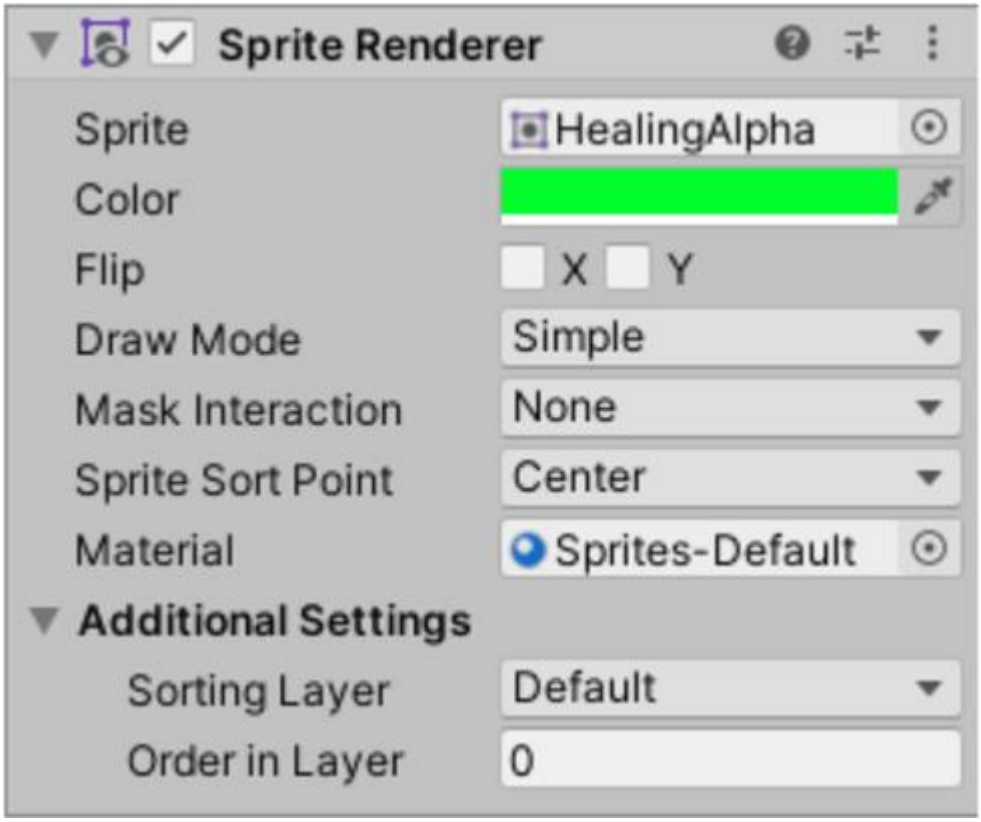

Рис. 2.11. Компонент «Sprite renderer»

Спроектовану анімацію гри розглянемо на прикладі фігури, для якої в проекті застосовано компонент «Animator» (рис. 2.12). Він зв’язує об’єкт з файлами анімацій (.anim), що створюються у вікні «Animation».

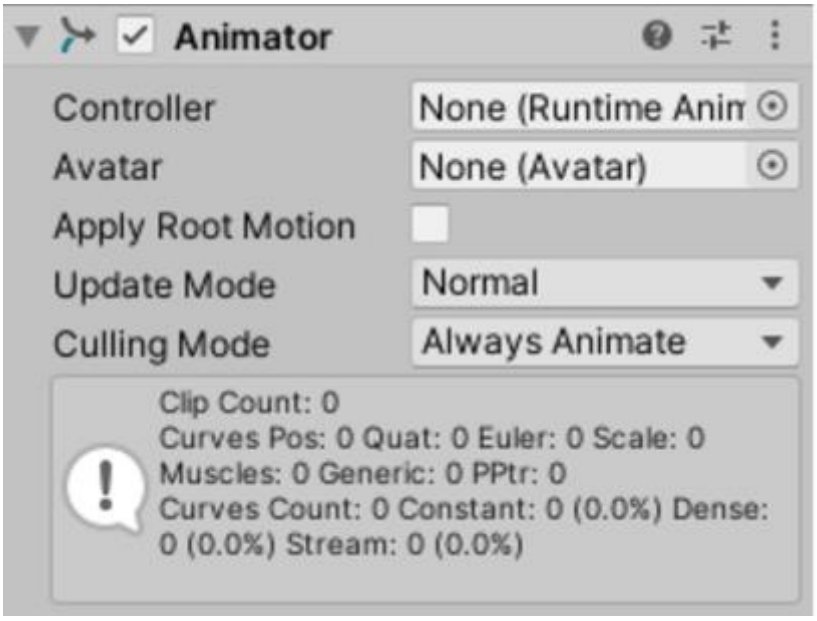

Рис. 2.12. Компонент «Animator»

Кожен файл .anim повинен містити одну дію, таким чином після створення всіх можливих дій об’єкта необхідно додати файли анімації до вікна «Animator» (рис. 2.13) і створити зв’язки між ними та налаштувати тригери переходу до наступної анімації для кожного зв’язка.

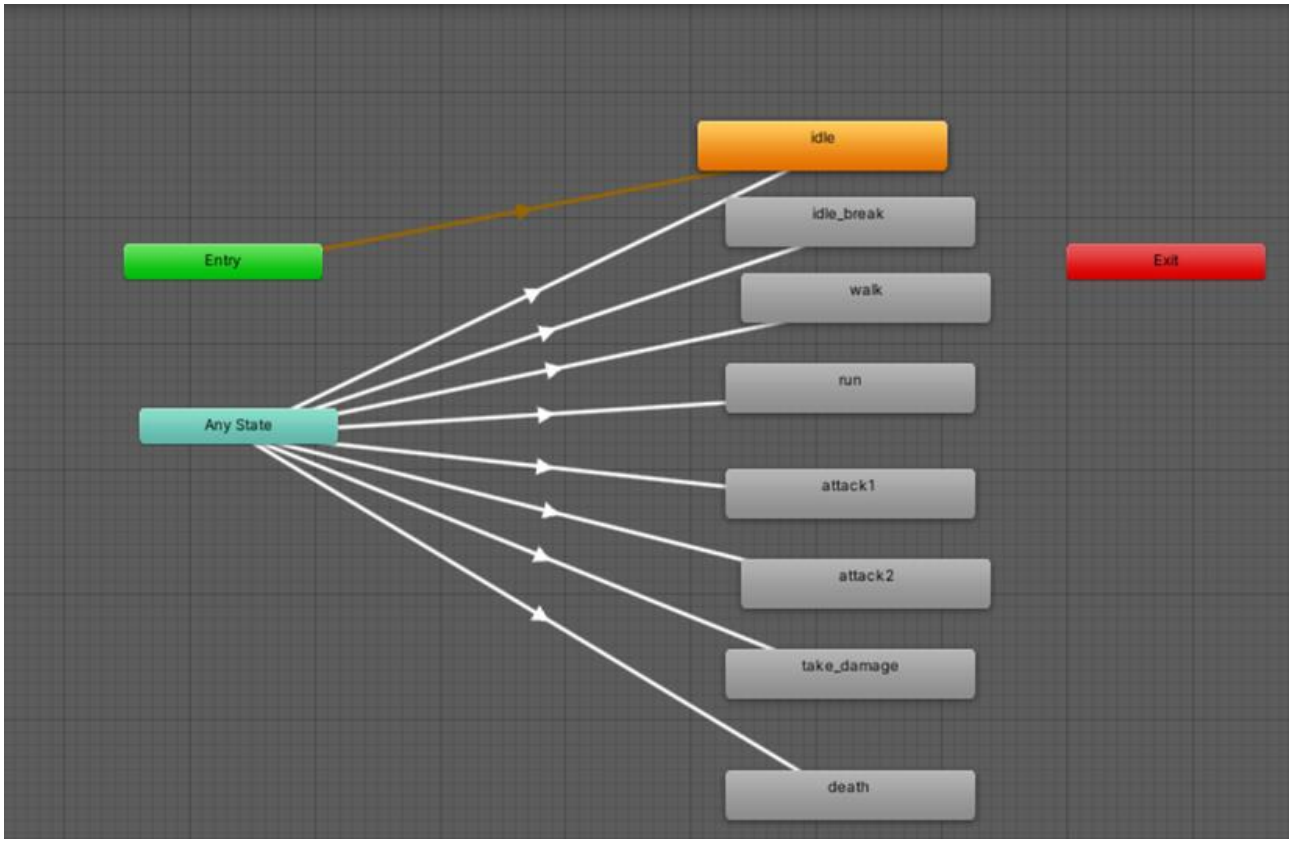

Рис. 2.13. Вікно «Animator»

Група компонентів «Collider » додає колізію, тобто фізичну взаємодію з іншими об’єктами. Розглянемо компонент «Box Collider» (рис. 2.14) – він не дає фігурі провалитися через куб ігрової дошки.

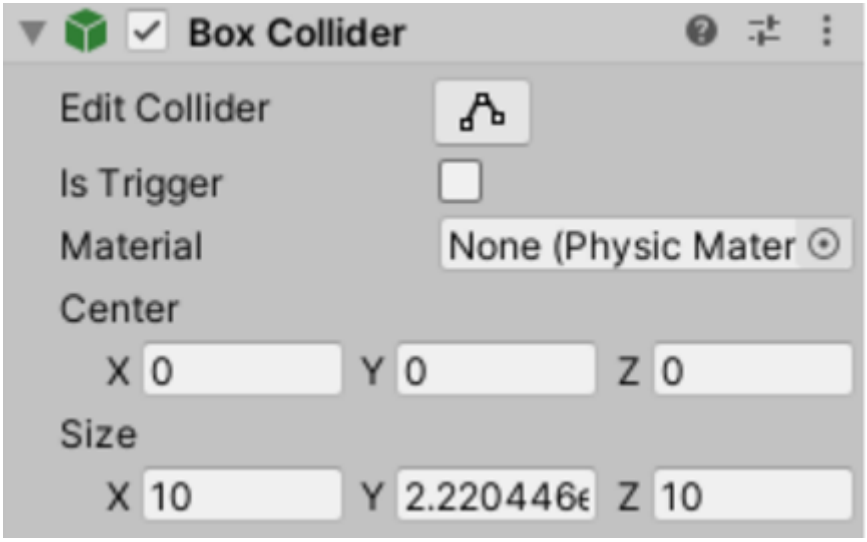

Рис. 2.14. Компонент «Rigidbody»

Створений користувачем компонент може керувати властивостями інших компонентів, додавати нові, змінювати ігрову сцену та використовувати всі можливості мови C#. Лістинг всіх користувацьких скриптів знаходиться у додатку А.

2.5.3. Реалізація ігрових елементів

Дизайн гри створений у стилі фентезі, що набирає чималої популярності серед сучасної молоді. Серед використаних засобів обрамлення містяться:

- знайомі та привабливі декорації жанру фентезі;
- цікаві та стилізовані персонажі;
- затишна, заспокійлива атмосфера.

Зовнішній вигляд інтерфейсу тісно пов’язаний із самою грою та сутністю ігрового процесу. Зокрема, старовинність гри вплинула на вибір стилю оточення: епохи Середньовіччя. Поштовхом до використання у купі із цим фентезі-декорацій стала значна популярність цього жанру серед аудиторії в наш час. Персонажі, що оточують гравця, створені та стилізовані відповідно до жанру. Приклади персонажів наведено на рис. 2.15.

Рис. 2.15. Персонажі гри

Саме ігрове поле, що буде знаходитись перед гравцем більшу частину ігрового процесу, також стилізоване під задану епоху, та має вигляд, власне, стола з листом паперу, на якому розкреслене ігрове поле, а гра ведеться фішками. Ліворуч та праворуч від поля наявні панелі з основним функціоналом. Усе графічне обрамлення було створене спеціально для гри.

Для елемента платформа створено наступні об’єкти і компоненти (табл. 2.1). Суть цієї платформи полягає у тому, щоб розміщувати фігури та розраховувати їх позиції. Вигляд і дія платформи відображені на рис. 2.16.

Таблиця 2.1

Компоненти ігрової дошки

Об’єкт	Компонент	Призначення
Spikes	Mesh renderer	Візуальна модель
	Box Collider	Модель колізії блоків
	Hill (Script)	Відображення партиклів
Platform	Mesh renderer	Модель колізії платформи
	Box Collider	Фізична модель платформи
Key	TextMeshPro	Відображення вибору платформи
Pos y	Transform	Позиція 1
Pos x	Transform	Позиція 2

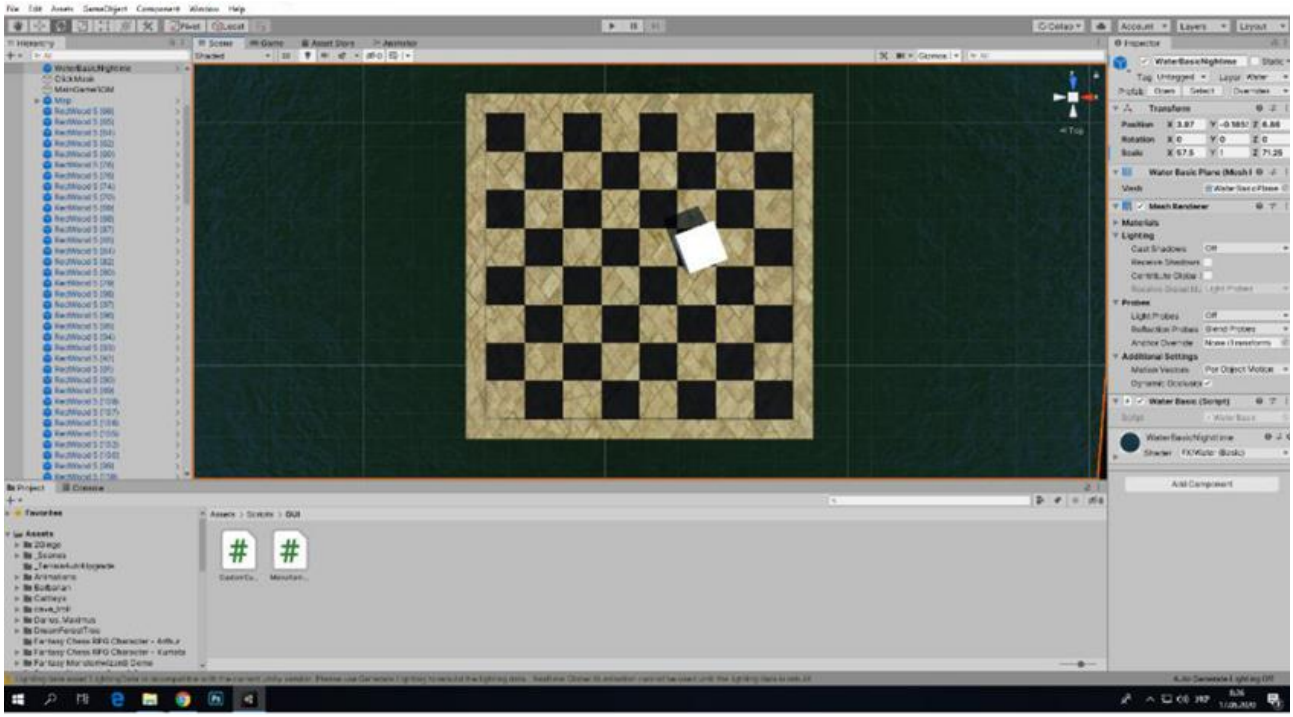

Рис. 2.16. Концепт ігрової дошки

Для елементу камера створено наступні об’єкти і компоненти (табл. 2.2) та основний скрипт «pRotated». Суть камери полягає у тому, щоб відображати ігрову сцену і зміни, які на ній відбуваються.

Таблиця 2.2

Властивості скрипта «Camera»

Властивість	Тип	Призначення
X	Float	Кординати камери по x
Y	Float	Кординати камери по y
Button	KeyCode	Кнопка зміни кадру
Key	TextMesh Pro	Відображення кнопки платформи
Властивість	Тип	Призначення
Platform	Transform	Координати платформи
CameraAnimator	Bool	Відображення заданої анімації

Для елементу фігура створено наступні об’єкти і компоненти (табл. 2.3).

Вигляд і дія фігури відображені на рис. 2.17.

Таблиця 2.3

Компоненти однієї з фігур

Об’єкт	Компонент	Призначення
Player	Chess (Script)	Управління фігурою
	Sprite Renderer	Візуальна модель фігури
	Box Collider	Модель колізії фігури
	Rigidbody 2D	Фізика фігури
	Animator	Анімація фігури
	Particle System	Система партиклів, які відображуються згідно стану фігури

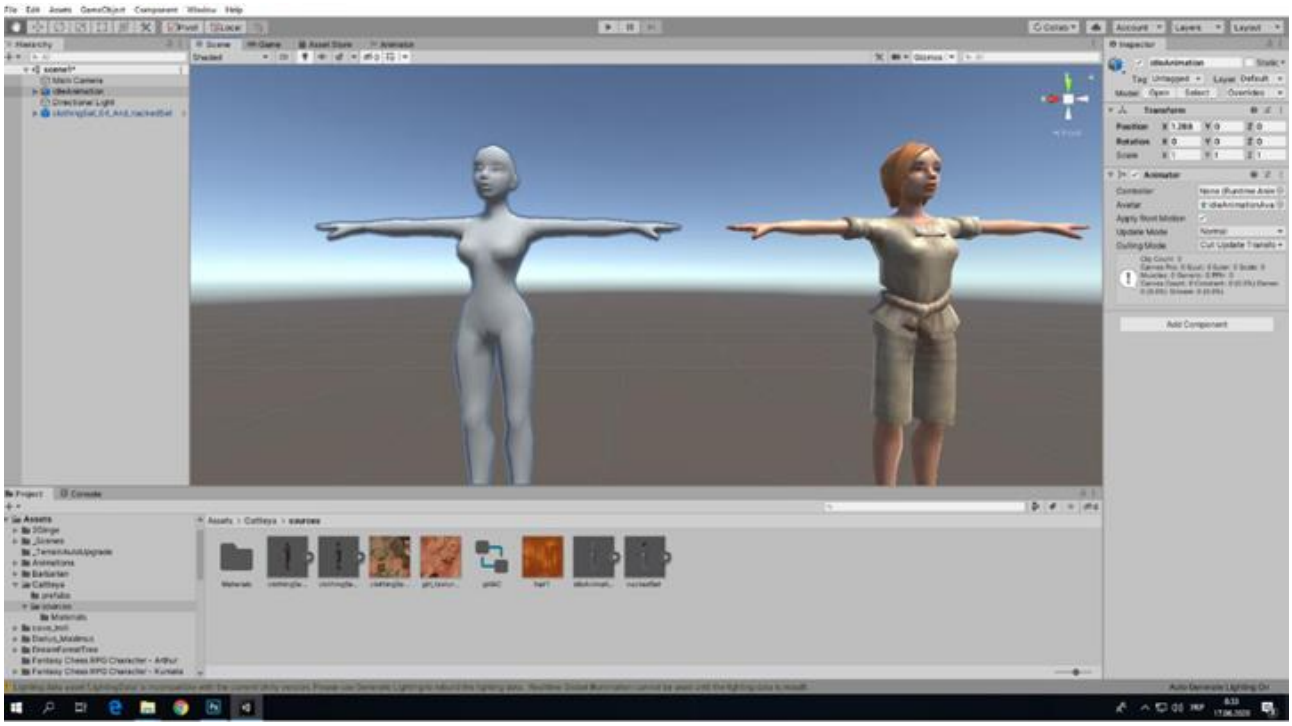

Рис. 2.17. Концепт ігрової фігури

Скрипт «Enemy» призначений для розпізнавання черги ходу гравця. Тобто коли хід першого гравця, то другий гравець не може взаємодіяти з фігурами поки перший гравець не закінчить свій хід, та навпаки. Властивості

скрипта «Enemy» наведено в табл. 2.4.

Таблиця 2.4

Властивості скрипта «Enemy»

Властивість	Тип	Призначення
Player	Player	Посилання на персонажа
Type	Float	Положення на дошці
King	Bool	Перевірка на наявність основної фігури
Result	Bool	Результат гравця

2.6. Обґрунтування та організація вхідних та вихідних даних програми

Згідно з постановкою завдання у якості вхідних даних програми виступають:

– вибір фігури та направлення ходу для неї;
– вибір сторони змагання;
– вибір режиму в грі.

Вихідними даними є:

– відображення фігур;
– програвання мелодії;
– відображення анімації;
– вивід результату гри.

2.7. Опис роботи розробленого програмного продукту

2.7.1. 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.

2.7.2. Використані програмні засоби

Реалізація виконана на платформі Unity, мова програмування для написання скриптів та ігрової логіки – C#. Об’єктно-орієнтована концепція мови програмування C# найліпше підходить для описання ігрової логіки об’єктів за допомогою системи класів. Вибране програмне забезпечення Blender цілком підійшло для моделювання тривимірних об’єктів, що складають рівні.

2.7.3. Виклик та завантаження програми

На початку роботи з грою, необхідно завантажити її у вигляді rar архіву.

Після завантаження архіву, необхідно розпакувати весь його зміст в комфортне для користувача місце. Після розпакування, натиснути подвійним кліком по Valhalla Auto Chess.exe для старту гри (рис. 2.18).

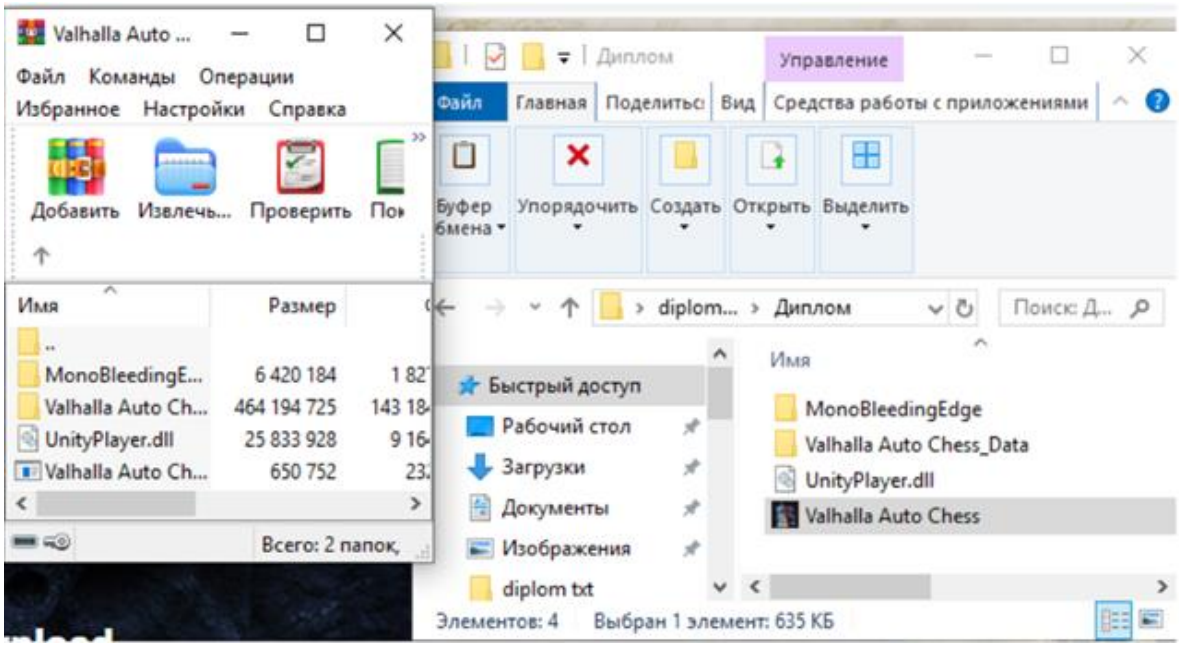

Рис. 2.18. Розпакування та запуск гри

2.7.4. Опис інтерфейсу користувача

Після запуску гри, перед користувачем з’являється Головне меню (рис. 2.19), яке представлене декількома кнопками взаємодії з проектом: грати, візуалізація гри, додаткові опції, вихід.

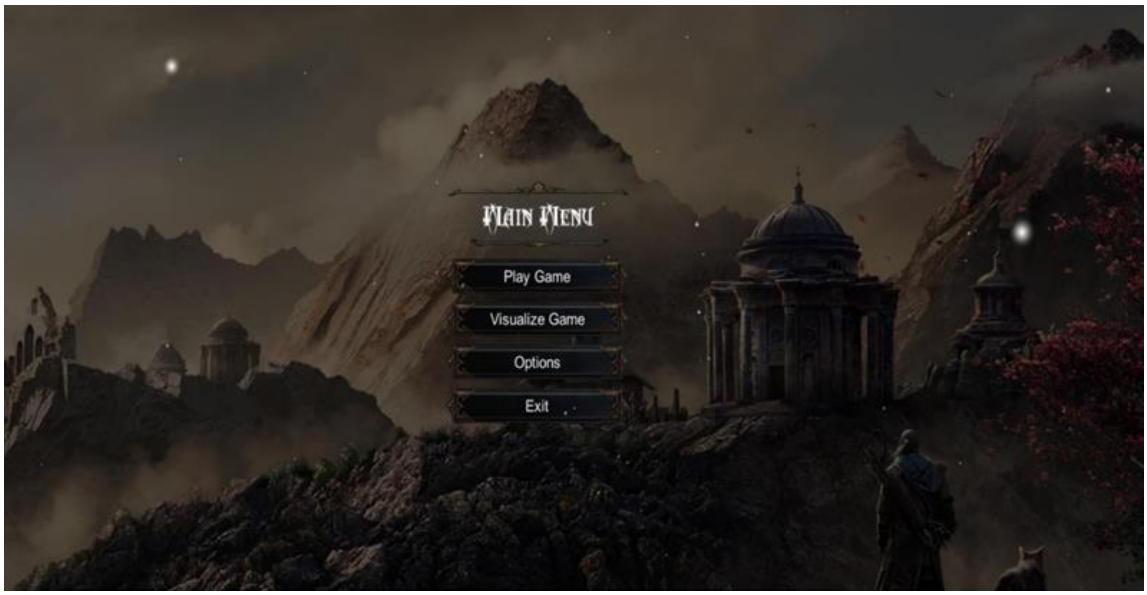

Рис. 2.19. Головне меню

При виборі кнопки Options користувач потрапляє в вікно редагування графіки та звуку (рис. 2.20-2.21).

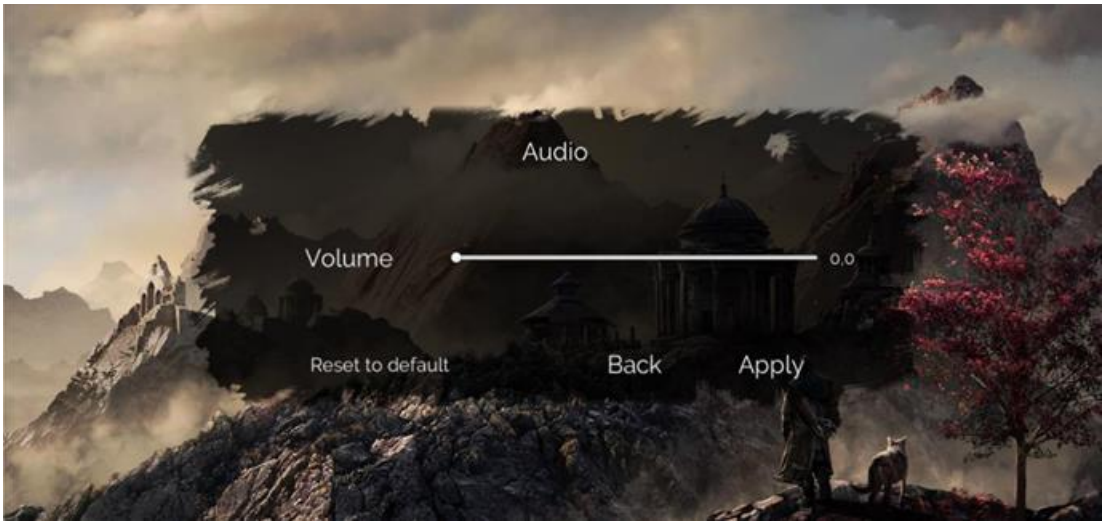

Рис. 2.20. Вікно налаштування музики

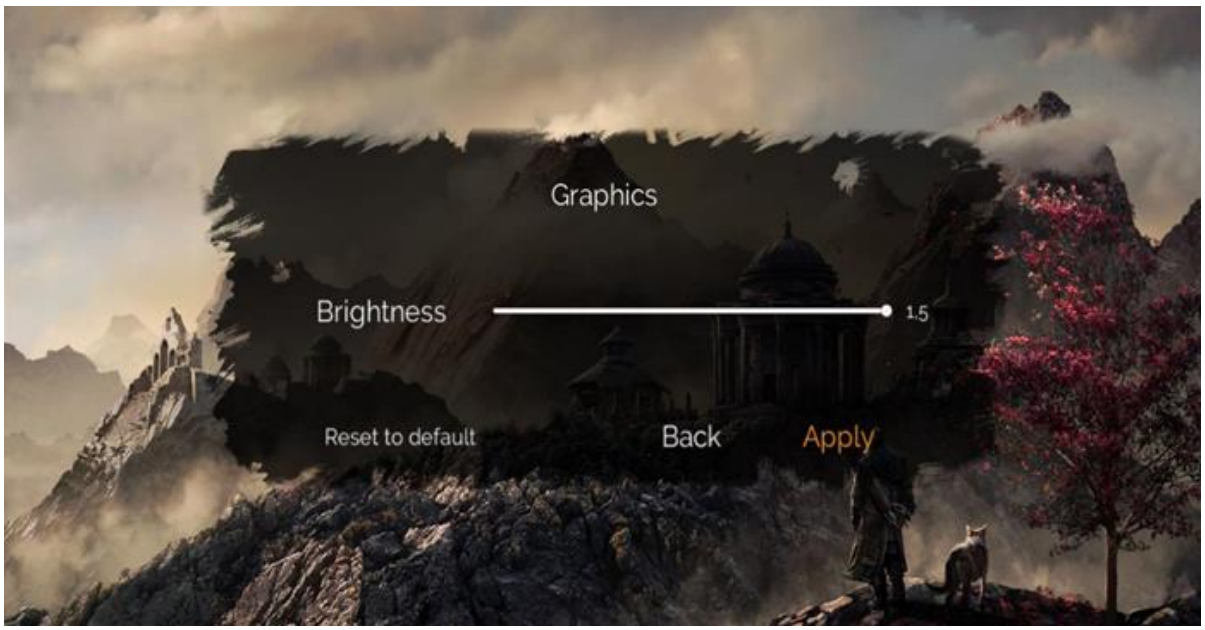

Рис. 2.21. Вікно налаштування графіки

При виборі кнопки Visualize Game користувача ознайомлюють з гейплеєм гри. Якщо натиснути на кнопку Exit, то відбувається вихід з гри. Коли користувач вибирає кнопку Play Game, він потрапляє в головну сцену, в якій завантажуються всі об’єкти, з якими він може взаємодіяти в процесі гри (рис. 2.22).

Рис. 2.23. Головна сцена гри

При завантаженні головної сцени, за допомогою написаних скриптів, виконується задана анімація камери (рис. 2.24) та завантаження всіх фігур на поле бою.

Рис. 2.24. Камера темної сторони

Після стабілізування камери можна побачити, що кожній фігурі задана анімація простою. У всіх фігур прописано індивідуальний стиль атаки. Переміщення відбувається по клітинкам в заданому просторі масиву чисел.

В процесі гри кожна з фігур має свою унікальну анімацію (рис. 2.25-2.27)

Рис. 2.25. Анімація фігури 1

Рис. 2.26. Анімація фігури 2

Рис. 2.27. Анімація фігури 3

На головній сцені розташовані дві кнопки, які допомагають взаємодіяти гравцеві з позицією камери (рис. 2.28-2.29). Це реалізовано, для підбору індивідуального кута сприйняття інформації.

Рис. 2.28. Зміна позиції камери (права сторона)

Рис. 2.29. Зміна позиції камери (ліва сторона)

Кожна фігура рухається за правилами шахових фігур. В програмі

передбачена допомога (візуальна підказка) для зображення можливих ходів юнітів (рис. 2.30).

Рис. 2.30. Підказка ходу юніту

На рис. 2.29 зображено ігровий процес.

Рис. 2.29. Ігровий процес

Партиклі або система частинок - це часто використовуваний у відеоіграх інструмент для подання об'ємних ефектів, які не мають чітких геометричних кордонів. На рис. 2.30 зображено приклад партикли ігрових подій, які були розроблені в даній грі.

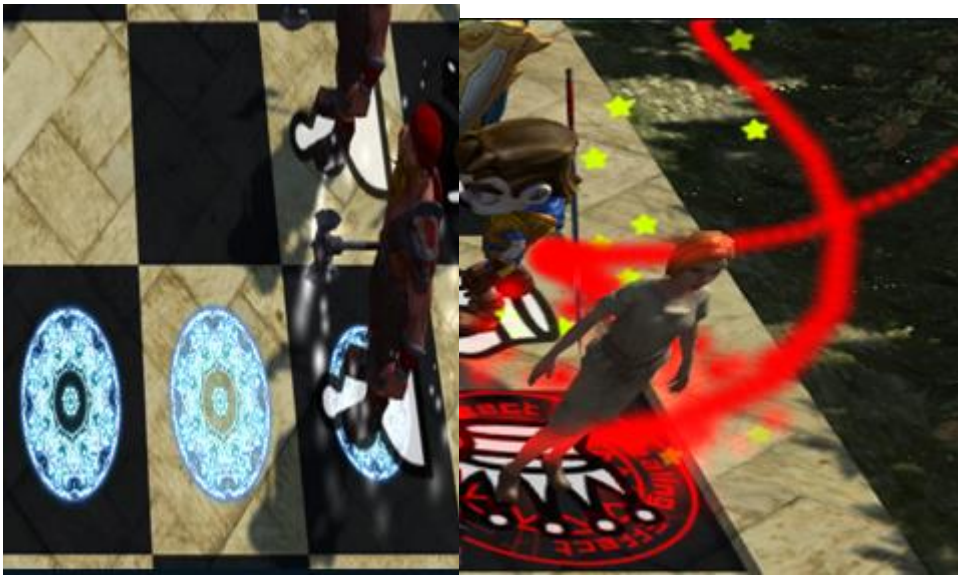

Рис. 2.30. Партиклі ігрового процесу 1

Рис. 2.31 Партиклі ігрового процесу 2

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 – 40;
- 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

$$Ç_{Çï} = 592 \cdot 40 = 23680, \text{грн.}$$

$З_{МВ}$ – 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.

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

$$\hat{E}_{ïï} = 23680 + 1419 = 25100 \text{ , грн.}$$

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 25100 грн.

ВИСНОВКИ

Завданням кваліфікаційної роботи є спроектувати та розробити стратегічну відео гру «Valhalla Auto Chess» на платформі Unity, метою якої є розвинути логіку та кмітливість гравців.

Переміщення фігур відбувається в тривимірному просторі, можна рухатися вліво, вправо, вперед та назад.

На розробленій карті відображається зміст назви гри, є контролер камери, щоб гравець зміг знайти для себе оптимальне положення, для більш чіткого сприйняття інформації. Ігрова дошка розрахована для двох гравців, тобто коли відбувається хід гравця, він може взаємодіяти тільки зі своїми фігурами.

При закінчені гри відображається переможець. В будь-який момент гра може бути призупинена для редагування користувачем графіки та гучності звуків.

Головне меню містить кнопки виходу з гри та ігрових налаштувань. Додатково, для ознайомлення з правилами гри та особливостями додатку, в головному меню міститься кнопка інструкції, яка запускає записаний гемплей.

Реалізація виконана на платформі Unity, мова програмування для написання скриптів та ігрової логіки – C#. Об’єктно-орієнтована концепція мови програмування C# найліпше підходить для описання ігрової логіки об’єктів за допомогою системи класів. Вибране програмне забезпечення Blender цілком підійшло для моделювання тривимірних об’єктів, що складають рівень.

Практичне значення додатку полягає в тому, що система може використовуватись користувачами для розважальних заходів та приємного відпочинку, а також тренування логічного мислення.

Актуальність розробленого проекту полягає в тому, що в роботі реалізовано на основі розробленого алгоритму прийняття рішень досконалу стратегію ведення гри та запропоновано працеспроможний кінцевий продукт для його використання для розваг та тренування логічного мислення, розвитку інтелектуальних, аналітичних здібностей за допомогою ігор шашкового типу.

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 25100 грн.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

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

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

3. Документація Visual Studio URL: docs.microsoft.com/ru-ru/visualstudio/get-started/ visual-studio-ide?view=vs-2019. дата звернення: 15.01.2021.

4. Розендаль Т. Інструменти моделювання в Blender / Тон Розендаль., 2016. – 135 с.

5. Чекмарьов А. Операційні системи / А. Чекмарьов., 2014. – 93 с.

6. Поташників А. Особливості застосування методів DirectX / А. Поташників., 2016. – 48 с.

7. Шашки // Игры и развлечения. Книга 2 / Сост. Л.М. Фирсова. - М.: Молодая гвардия, 1990. - С. 89 -116. - 234 с.

8. В.Пак. Популярный шашечный практикум..Москва: АСТ; Донецк: Сталкер ,2004.- 285 с

9. А.Вирный. Немного о шашках, но по существу.- Москва: ФАИР-ПРЕСС,2005.- 328 с.-ISBN 5-8183-0674-7

10. Голосуев В.М. Древняя и загадочная игра. Мир шашек. - СПб.: Интеграф, 1997. - 216 с.

11. 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).

12. 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.

13. 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: 15.01.2021.

14. 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.

15. 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.

16. 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.

17. Методичні рекомендації до виконання кваліфікаційних робіт бакалаврів напряму підготовки 121 «Програмна інженерія» галузі знань 12 Інформаційні технології/, Л.М. Коротенко , О.С. Шевцова; Нац. гірн. ун-т. – Д : ДВНЗ НГУ, 2019. – 65 с.

18. Методичні рекомендації щодо написання, оформлення та представлення учнівських науково-дослідницьких робіт учнів – членів Малої академії наук України / Г.Г. Півняк, Л.М. Коротенко, І.М. Удовик, Є.М. Головня – Д.: ДВНЗ «Національний гірничий університет», 2017. – 24 с.

19. 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).

20. Ухлинов А. М. Управление безопасностью информации в автоматизированных системах. М.: МИФИ, 1996. – 278 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;
}
}

using UnityEngine;
using System.Collections;
using System;
using System.Collections.Generic;
public class King : Chessman {
public override string Annotation()
{
return "K";
}
public override bool[,] PossibleMove()
{
bool[,] moves = new bool[8, 8];
int[] dx = { -1, 0, 1, 1, 1, 0, -1, -1 };
int[] dy = { 1, 1, 1, 0, -1, -1, -1, 0 };
int x, y;
bool[,] enemyMoves = GetAllEnemyMoves(BoardManager.Instance.GetAllChessmans());
for (int i = 0; i < 8; i++)
{
x = CurrentX + dx[i];
y = CurrentY + dy[i];
if (x < 0 || x > 7 || y < 0 || y > 7 || enemyMoves[x, y] == true)
continue;
Chessman c = BoardManager.Instance.Chessmans[x, y]; if (c == null)
moves[x, y] = true;
else if (c.isWhite != isWhite)
moves[x, y] = true;
}
return moves;
}
private bool[,] GetAllEnemyMoves(List<GameObject> enemies)
{
List<bool[,]> listMoves = new List<bool[,]>(); foreach (GameObject go in enemies) {
if (go != null)
```

```
{
Chessman chessman = go.GetComponent<Chessman>(); if (chessman.isWhite != isWhite) {
listMoves.Add(chessman.PossibleEat());
}
}
}
bool[,] result = new bool[8, 8];
for (int i = 0; i < 8; i++)
{
for (int j = 0; j < 8; j++)
{
foreach (bool[,] moves in listMoves)
{
if (moves[i, j])
{
result[i, j] = true;
break;
}
}
}
}
return result;
}
public override bool[,] PossibleEat()
{
bool[,] moves = new bool[8, 8];
int[] dx = { -1, 0, 1, 1, 1, 0, -1, -1 };
int[] dy = { 1, 1, 1, 0, -1, -1, -1, 0 };
int x, y;
for (int i = 0; i < 8; i++)
{
x = CurrentX + dx[i];
y = CurrentY + dy[i];
if (x < 0 || x > 7 || y < 0 || y > 7)
continue;
Chessman c = BoardManager.Instance.Chessmans[x, y]; if (c == null)
moves[x, y] = true;
else if (c.isWhite != isWhite)
moves[x, y] = true;
}
return moves;
}
}

using UnityEngine;
using System.Collections;
using System;
public class Knight : Chessman {
public override string Annotation()
{
return "N";
}
```

```
public override bool[,] PossibleEat()
{
return PossibleMove();
}
public override bool[,] PossibleMove()
{
bool[,] moves = new bool[8, 8];
// UpLeft
KnightMove(CurrentX - 1, CurrentY + 2, ref moves);
// UpRight
KnightMove(CurrentX + 1, CurrentY + 2, ref moves);
// RightUp
KnightMove(CurrentX + 2, CurrentY + 1, ref moves);
// RightDown
KnightMove(CurrentX + 2, CurrentY - 1, ref moves);
// DownLeft
KnightMove(CurrentX - 1, CurrentY - 2, ref moves);
// DownRight
KnightMove(CurrentX + 1, CurrentY - 2, ref moves);
// LeftUp
KnightMove(CurrentX - 2, CurrentY + 1, ref moves);
// DownRight
KnightMove(CurrentX - 2, CurrentY - 1, ref moves);
return moves;
}
private void KnightMove(int x, int y, ref bool[,] moves)
{
if (x >= 0 && x < 8 && y >= 0 && y < 8)
{
Chessman c = BoardManager.Instance.Chessmans[x, y]; if (c == null)
moves[x, y] = true;
else if (c.isWhite != isWhite)
moves[x, y] = true;
}
}
}

using UnityEngine;
using System.Collections;
using System;
public class Pawn : Chessman {
public override string Annotation()
{
return "P";
}
public override bool[,] PossibleEat()
{
bool[,] moves = new bool[8, 8];
Chessman c1;
//        white team moves if (isWhite)
{
// Diagonal left
```

```
if (CurrentX != 0 && CurrentY != 7)
{
c1 = BoardManager.Instance.Chessmans[CurrentX - 1, CurrentY + 1];
if (c1 == null)
{
moves[CurrentX - 1, CurrentY + 1] = true;
}
}
// Diagonal right
if (CurrentX != 7 && CurrentY != 7)
{
c1 = BoardManager.Instance.Chessmans[CurrentX + 1, CurrentY + 1];
if (c1 == null)
{
moves[CurrentX + 1, CurrentY + 1] = true;
}
}
}
else
{
// Diagonal left
if (CurrentX != 0 && CurrentY != 0)
{
c1 = BoardManager.Instance.Chessmans[CurrentX - 1, CurrentY - 1]; if (c1 == null)
{
moves[CurrentX - 1, CurrentY - 1] = true;
}
}
// Diagonal right
if (CurrentX != 7 && CurrentY != 0)
{
c1 = BoardManager.Instance.Chessmans[CurrentX + 1, CurrentY - 1];
if (c1 == null)
{
moves[CurrentX + 1, CurrentY - 1] = true;
}
}
}
return moves;
}
public override bool[,] PossibleMove()
{
bool[,] moves = new bool[8, 8];
int[] e = BoardManager.Instance.EnPassantMove; Chessman c1, c2;
//        white team moves if (isWhite)
{
// Diagonal left
if (CurrentX != 0 && CurrentY != 7)
{
if (e[0] == CurrentX - 1 && e[1] == CurrentY + 1)
moves[CurrentX - 1, CurrentY + 1] = true;
c1 = BoardManager.Instance.Chessmans[CurrentX - 1, CurrentY + 1];
```

```
if (c1 != null && !c1.isWhite)
{
moves[CurrentX - 1, CurrentY + 1] = true;
}
}
// Diagonal right
if (CurrentX != 7 && CurrentY != 7)
{
if (e[0] == CurrentX + 1 && e[1] == CurrentY + 1)
moves[CurrentX + 1, CurrentY + 1] = true;
c1 = BoardManager.Instance.Chessmans[CurrentX + 1, CurrentY + 1];
if (c1 != null && !c1.isWhite)
{
moves[CurrentX + 1, CurrentY + 1] = true;
}
}
// Middle
if (CurrentY != 7)
{
c1 = BoardManager.Instance.Chessmans[CurrentX, CurrentY + 1];
if (c1 == null)
{
moves[CurrentX, CurrentY + 1] = true;
}
}
//        Middle on first move if (CurrentY == 1)
{
c1 = BoardManager.Instance.Chessmans[CurrentX, CurrentY + 1];
c2 = BoardManager.Instance.Chessmans[CurrentX, CurrentY + 2];
if (c1 == null && c2 == null)
{
moves[CurrentX, CurrentY + 2] = true;
}
}
} else
{
//        Diagonal left
if (CurrentX != 0 && CurrentY != 0)
{
if (e[0] == CurrentX - 1 && e[1] == CurrentY + 1)
moves[CurrentX - 1, CurrentY - 1] = true;
c1 = BoardManager.Instance.Chessmans[CurrentX - 1, CurrentY - 1]; if (c1 != null &&
c1.isWhite)
{
moves[CurrentX - 1, CurrentY - 1] = true;
}
}
// Diagonal right
if (CurrentX != 7 && CurrentY != 0)
{
if (e[0] == CurrentX + 1 && e[1] == CurrentY - 1)
moves[CurrentX + 1, CurrentY - 1] = true;
```

```
c1 = BoardManager.Instance.Chessmans[CurrentX + 1, CurrentY - 1];
if (c1 != null && c1.isWhite)
{
moves[CurrentX + 1, CurrentY - 1] = true;
}
}
// Middle
if (CurrentY != 0)
{
c1 = BoardManager.Instance.Chessmans[CurrentX, CurrentY - 1];
if (c1 == null)
{
moves[CurrentX, CurrentY - 1] = true;
}
}
//        Middle on first move if (CurrentY == 6)
{
c1 = BoardManager.Instance.Chessmans[CurrentX, CurrentY - 1];
c2 = BoardManager.Instance.Chessmans[CurrentX, CurrentY - 2];
if (c1 == null && c2 == null)
{
moves[CurrentX, CurrentY - 2] = true;
}
}
}
return moves;
}
}

using UnityEngine;
using System.Collections;
using System;
public class Queen : Chessman {
public override string Annotation()
{
return "Q";
}
public override bool[,] PossibleEat()
{
return PossibleMove();
}
public override bool[,] PossibleMove()
{
bool[,] moves = new bool[8, 8];
Chessman c;
int i, j;
// Right
i = CurrentX;
while (true)
{
i++;
if (i >= 8)
```

```
break;
c = BoardManager.Instance.Chessmans[i, CurrentY];
if (c == null)
{
moves[i, CurrentY] = true;
}
else
{
if (c.isWhite != isWhite)
moves[i, CurrentY] = true;
break;
}
}
// Left
i = CurrentX;
while (true)
{
i--;
if (i < 0)
break;
c = BoardManager.Instance.Chessmans[i, CurrentY];
if (c == null)
{
moves[i, CurrentY] = true;
}
else
{
if (c.isWhite != isWhite)
moves[i, CurrentY] = true;
break;
}
}
// Up
i = CurrentY;
while (true)
{
i++;
if (i >= 8)
break;
c = BoardManager.Instance.Chessmans[CurrentX, i]; if (c == null)
{
moves[CurrentX, i] = true;
}
else
{
if (c.isWhite != isWhite)
moves[CurrentX, i] = true;
break;
}
}
// Down
i = CurrentY;
```

```
while (true)
{
i--;
if (i < 0)
break;
c = BoardManager.Instance.Chessmans[CurrentX, i];
if (c == null)
{
moves[CurrentX, i] = true;
}
else
{
if (c.isWhite != isWhite)
moves[CurrentX, i] = true;
break;
}
}
//        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 class Rook : Chessman {
public override string Annotation()
{
return "R";
}
public override bool[,] PossibleEat()
{
return PossibleMove();
}
public override bool[,] PossibleMove()
{
bool[,] moves = new bool[8, 8];
Chessman c;
// Right
int i = CurrentX;
while (true)
{
i++;
```

```
if (i >= 8)
break;
c = BoardManager.Instance.Chessmans[i, CurrentY]; if (c == null)
{
moves[i, CurrentY] = true;
} else
{
if (c.isWhite != isWhite) moves[i, CurrentY] = true;
break;
}
}
// Left
i = CurrentX;
while (true)
{
i--;
if (i < 0)
break;
c = BoardManager.Instance.Chessmans[i, CurrentY];
if (c == null)
{
moves[i, CurrentY] = true;
}
else
{
if (c.isWhite != isWhite)
moves[i, CurrentY] = true;
break;
}
}
// Up
i = CurrentY;
while (true)
{
i++;
if (i >= 8)
break;
c = BoardManager.Instance.Chessmans[CurrentX, i];
if (c == null)
{
moves[CurrentX, i] = true;
}
else
{
if (c.isWhite != isWhite)
moves[CurrentX, i] = true;
break;
}
}
// Down
i = CurrentY;
while (true)
```

```
{
i--;
if (i < 0)
break;
c = BoardManager.Instance.Chessmans[CurrentX, i]; if (c == null)
{
moves[CurrentX, i] = true;
}
else
{
if (c.isWhite != isWhite)
moves[CurrentX, i] = true;
break;
}
}
return moves;
}
}
```

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