ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра комп‘ютерної інженерії

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «**Розробка гри "The loss of Bagrash" в жанрі «покрокова стратегія» на двигуні Unity»**

Виконав: студент 4 курсу, групи ПД–43
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
Музика А.П
(прізвище та ініціали)
Керівник Дібрівний О.А.
(прізвище та ініціали)
Рецензент _____
(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут Телекомунікацій

Кафедра Телекомунікаційних технологій
Ступінь вищої освіти - «Бакалавр»
Напрям підготовки -121 – "Інформаційні технології"

ЗАТВЕРДЖУЮ
Завідувач кафедри
Телекомунікаційних технологій
А.В. Бондаренко
“____” _____2021 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

_____
(прізвище, ім‘я, по батькові)

1.Тема роботи: «Розробка гри "The loss of Bagrash" в жанрі покрокова стратегія на двигуні Unity»
Керівник роботи Дібрівний О.А., доктор філософії
(прізвище, ім‘я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від «18» лютого 2022 року № _____.
2. Строк подання студентом роботи «3» червня 2022 року
3. Вхідні дані до роботи:
Ігри в жанрі «покрокова стратегія»
Науково-технічна література з питань, пов’язаних з програмним забезпеченням, щодо розробки ігор на двигуні Unity; ;

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).
4.1 Вплив ігор на психологію людини і придбання життєвих навичок.
4.2 Розробка структури додатку та технології оптимізації.
4.3 Програмна реалізація.
4.4 Огляд використаних інструментів для створення ігор.

5. Перелік демонстраційного матеріалу (назва основних слайдів)
1. Актуальність проблеми
2. Огляд аналогів.
3. Мета, об'єкт, предмет.
4. Технічне завдання.

5. Засоби розробки.
6. Етапи проектування та розробки.
7. Екранні форми.
8. Апробація.
9. Висновки.

6. Дата видачі «11» квітня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	11.04.22	Виконано
2	Розділ 1. Вплив ігор на психологію людини і придбання життєвих навичок	19.04.22	Виконано
3	Розділ 2. Розробка структури додатку та технології оптимізації.	25.04.22	Виконано
4	Розділ 3. Програмна реалізація	10.05.22	Виконано
5	Розділ 4. Огляд використаних інструментів для створення ігор	17.05.22	Виконано
6	Вступ, висновки, реферат	20.05.22	Виконано
7	Розробка обов‘язкових демонстраційних матеріалів	26.05.22	Виконано
8	Попередній захист роботи	31.05.22	
9	Здача роботи	3.06.22	

Студент _____ Музика А.П.
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи: 42 с., 19 рис., 2 дод., 18 джерела.

Мета роботи – підвищення зацікавленості гравців в ігровому процесі покрокової стратегії за рахунок реалізації локального багатокористувацького режиму(підтримкою керування кількома ігровими акаунтами).

Об`єкт дослідження – ігровий процес в жанрі «покрокова стратегія».

Предмет дослідження – програмне забезпечення для реалізації механіки ігор в жанрі «покрокових стратегій».

Створити гру на двигуні Unity в жанрі «покрокова стратегія» в якій двоє гравців мають можливість зіграти один проти одного на одному персональному комп'ютері, для цього потрібно виконати наступні завдання:

Розробити систему генерації сітки для бою та переміщення.

Впровадити взаємодію гравця з ігровою сіткою.

Реалізувати спосіб переміщення ігрових фігур по сітці.

Розробити юнітів з різним варіантами атаки.

Розробити меню.

Змоделювати візуальне середовище.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ПК – персональний комп’ютер.

TBS (Turn-based Strategy) - покрокова стратегія.

RPG (Role-Playing Game) – жанр відеоігор, де основна частина ігрового процесу полягає в управлінні персонажем чи групою персонажів, які досліджують ігровий світ, виконують різноманітні завдання та розвиваються, слідуючи сюжету.

Ассети – це компоненти, які є графікою, звуковим супроводом або скриптами. Вони прикріплюються до об'єктів та становлять важливу частину гри.

UML (Unified Modeling Language) – уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування.

OOP (Object-oriented programming) – одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою.

IDE (Integrated Development Environment) – Інтегроване середовище розробки.

ЦП (Центральний процесор) – електронний блок або мікросхема, що виконує машинні інструкції (код програм).

API (Application Programming Interface) – програмний інтерфейс програми.

LOD (Levels Of Detail) – прийом у програмуванні тривимірної графіки, що полягає у створенні декількох варіантів одного об'єкта з різними ступенями деталізації, які перемикаються в залежності від видалення об'єкта від віртуальної камери.

CAD (Computer-Aided-Design) – автоматизована система, є організаційно-технічною системою, призначеною для автоматизації процесу проектування, що складається з персоналу та комплексу технічних, програмних та інших засобів автоматизації його діяльності .

GPU (Graphics Processing Unit) – графічний процесор, окремий пристрій персонального комп'ютера або ігрової приставки, виконує графічний рендеринг.

Рендеринг (Rendering) – комп'ютерна візуалізація, процес отримання зображення за моделлю з допомогою комп'ютерної програми.

ВСТУП

Обґрунтування вибору теми і її актуальність – покрокові стратегії навчають планувати дії на перед.

За короткий час в Україні, як і в інших країнах СНД, відбулися глибокі зміни в різних сферах життя. З нововведень, що змінили стиль життя та й інтереси осіб різного віку, можна виокремити комп'ютеризацію. Крім практичного використання в роботі та побуті, спостерігається захоплення комп'ютерними іграми, що швидко поширюється, особливо серед підлітків і дітей. У зв'язку з цим комп'ютерні ігри потрібно розглядати як своєрідний соціально-психологічний феномен, що сьогодні посідає все помітніше місце в житті людини.

Давно відомо, що в процесі розвитку мислення та формуванні особистості важливу роль відіграє не тільки освітній фактор, але й характер ігрової діяльності. Для дитини гра є провідною діяльністю, тому що саме під час гри вона засвоює значення і способи вживання предметів, а також різні варіанти соціальних відносин. Психологічну теорію гри розробляли Л. Виготський, С. Рубінштейн, Д. Ельконін та інші.

В ігровій діяльності підліток не тільки заміщає реальні предмети, але й приймає на себе ту чи іншу роль і починає діяти відповідно з нею. Роль у сюжетній грі полягає саме в тому, щоб виконувати обов'язки, що покладаються нею, і реалізовувати права стосовно інших учасників гри. Досвід ігрових та реальних взаємин у сюжетно-рольовій грі лягає в основу особливої властивості мислення, що дозволяє передбачити майбутню поведінку людей і залежно від цього будувати власну поведінку.

Гра є тим видом діяльності, що сприяє практичному освоєнню реального соціального простору - у символічних діях і заміщеннях дитина грає варіанти відносин людей, символічно ідентифікуючи себе з одними персонажами та відокремлюючи від інших.

Ігрова діяльність впливає на формування довільних психічних процесів. Умови гри вимагають зосередженості на змісті дій і сюжеті, на діючих особах чи

предметах, включених до ігрової ситуації. Ігрова ситуація впливає на мислення і психіку дитини, підлітка і дорослого. Гра сприяє розвиткові рефлексії, оскільки в цьому процесі виникає реальна можливість контролювати, як виконується будь-яка дія, що входить у процес спілкування. У рольовій грі формується здатність осмислювати свої власні дії, передбачати реакцію інших людей.

Ступінь вивчення проблеми:

В двадцять другому столітті комп’ютерні стратегії мають меншу популярність ніж двадцять чи тридцять років тому так і по сьогодні вони займають вагому нішу в геймдеві, перші представники були, є і на цей час являються адаптацією настільних ігор наприклад, шахи чи шашки. Ключовою особливістю TBS почерговий геймплей тобто зазвичай гравець має необмежений час на свій хід, а противник не в змозі прийняти ніякі дій доки не наступить черга його ходу. Ігри такого жанру неквапливі і потребують грамотного продумування своїх і ворожих ходів наперед. Поле для гри розмежовано квадратами або шестигранниками на яких здійснюються ходи розміщуються фігури та об'єкти учасників гри.

Об'єкт дослідження – ігровий процес в жанрі «покрокова стратегія».

Предмет дослідження – програмне забезпечення для реалізації механіки ігор в жанрі «покрокових стратегій».

Метою роботи є - підвищення зацікавленості гравців в ігровому процесі покрокової стратегії за рахунок реалізації локального багатокористувацького режиму(підтримкою керування кількома ігровими акаунтами).

Завдання роботи – розробка гри в жанрі «покрокова стратегія» на двигуні Unity.

Методика дослідження:

Основними методиками дослідження є збір інформації, вивчення джерел, експериментальні-ігрові методи, аналіз та обробка даних отриманих під час дослідження. Обрано найбільш відповідні технології для реалізації продукту, з використанням сучасних розробок в області розробки ігрових застосунків, з розділенням внутрішнього функціоналу на модулі відділені один від одного для найбільш ефективної розробки з виключенням помилок та підтримки продукту.

Зважаючи на висновки проведені при аналізі, було вирішено використати для розробки ігрового застосунку об’єктно орієнтовану мову програмування C# з використанням крос-платформової технології .NET Framework у середовищі розробки Visual Studio та ігровий рушій Unity, що виступає як інструмент для розробки відеоігор і застосунків.

Наукова новизна роботи – наукова новизна полягає в створенні гри в обраному жанрі з можливістю гри вдвох на одному ПК.

Практична значущість результатів:

Практично продукт може бути застосовано при наявності лише одного ПК та двох користувачів, що бажають зіграти один проти одного з єдиного пристрою.

1 ВПЛИВ ІГОР НА ПСИХОЛОГІЮ ЛЮДИНИ І ПРИДБАННЯ ЖИТТЄВИХ НАВИЧОК

В останні десятиліття поступово все більшу частку ігор складають ігри з комп’ютером, в яких людина реалізує діяльність щодо керування різними процесами ігрового простору. Вплив комп’ютерних ігор на поведінку людей не викликає сумніву.

Комп’ютерні ігри для багатьох дітей і підлітків стали важливішими за спілкування з однолітками, а для багатьох дорослих – цікавішими, ніж телебачення і книги. Про негативні наслідки "ігроманії" та особливості поведінки комп’ютерного гравця є вже навіть літературні твори, наприклад, "Принц Госплана".

Розглянемо ж деякі позитивні навички, що можуть формуватися у процесі комп’ютерних ігор.

Комп’ютерні ігри це не просто ще один вид діяльності, це діяльність конструювання світів. Конструювання світів є процесом творення образу світу в людській психіці. Всесвіт, планети, континенти, епохи, люди, різні істоти й техніка створюються, розвиваються та руйнуються. Щоб свідомість змогла охопити все це, всесвіт повинен бути згорнутий у "сконструйований світ".

Як сконструйований світ, будь-яка популярна комп’ютерна гра має власну фізику та властивості простору, штучну історію і перебіг часу, оригінальну філософію, етику і мораль. Гра дає гравцеві можливість активно діяти в сконструйованому світі. Це "... схоже на карнавал, але тільки ступінь волі "зміни масок" у комп’ютерних іграх незмірно вищий".

Початком епохи комп’ютерних ігор можна вважати появу "Space War", розроблену С. Расселом 1962 року в Массачусетському технологічному інституті. Але висока вартість комп’ютерів не дозволяли тоді цій грі стати масовим захопленням. Першу доступну усім комп’ютерну гру розробив Н. Бушель. Вона з’явилася в США 1972 року і називалася "Понг" (електронна версією пінг-понгу). Після цього з’явилося безліч ігор, які, власне кажучи, нічим не відрізнялися від

"Понга", але мали інше зовнішнє оформлення, наприклад, електронний хокей, футбол тощо. Однак знадобилося ще близько десяти років, аби комерційні комп'ютерні ігри дійшли у своєму розвитку до рівня складності "Космічної війни".

Справа в тому, що гра є розвагою для користувача, а для виробника це необхідність використання новітніх досягнень у галузі техніки, зокрема, у графіці, програмуванні складних видів руху та спеціальних ефектів тощо.

Навіть сьогодні тільки деякі прикладні програми, що використовуються більшістю користувачів, є такими ж ресурсномісткими і вибагливими до устаткування, як ігри. Особливо це стосується відео-плати, процесора, материнської плати і монітора. Для 97% завдань, що їх виконує домашній комп'ютер, немає жодної необхідності у надсучасних і дорогих системах. Найвимогливішими до устаткування залишаються саме ігри, хоча останнім часом розширення завдань, що ставляться перед домашнім комп'ютером, поступово додає в список програмного забезпечення і нові розділи, наприклад, відеофільми.

Існує кілька варіантів класифікації комп'ютерних ігор. Усі вони умовні, оскільки з'являється безліч ігор, що поєднують у собі елементи кожної категорії. Один з найпоширеніших варіантів класифікації має такий вигляд:

ігри типу "action", у тому числі і "RPG";

ігри пригодницькі, типу "quest";

ігри стратегічні;

ігри, що імітують транспорт;

віртуальне казино.

Ігри типу "action". 1994 року, відразу після своєї появи, набула популярності гра "Doom". Вона посідала перші місця у рейтингах, одержувала нагороди і стала джерелом ідей для цілої низки подібних ігор. Багато з них у свою чергу стали бестселерами, наприклад, "Doom Ultimate", "Quake" (від 1 до 3), "Hexen", "Unreal", "Duke Nukem 3D" (усі випуски), "Half-Life", "Serious Sam" (1 і 2) тощо.

Існує ряд особливостей, характерних для ігор сімейства "Doom". Одна з основних – це наявність багаторівневого лабіринту з пастками. Інша – обов'язкова наявність небезпечних монстрів. І, звичайно ж, герой, з яким ідентифікується

гравець. Основна ідея гри – припущення, що людина може пройти лабіринт, подолати монстрів і досягти мети.

"Doom"-подібні ігри постійно використовують архетип чудовиська. Їх світи населяють мерці, що оживають, демони, інопланетні чудовиська, агресивні роботи. Взагалі, розумні нелюдські істоти – традиційний елемент культури. Він реалізовувався в усній і писемній творчості, пізніше – у кінематографі, зараз – у віртуальних світах. Знайомство з монстрами починається в дитинстві, при читанні казок, міфічних оповідей. Пізніше, коли людина дорослішає, чудовиська, якими вона себе лякає, стають небезпечнішими (наприклад у романах С. Кінга та фільмах жахів). Сьогодні це, все частіше, чудовиська з комп'ютерних ігор.

Крім подібних "Doom" ігор до цього розділу можна зарахувати ігри військового характеру. У них, як правило, використовується як основа реальний чи близький до реального історичний простір. Прикладом таких ігор можуть слугувати "Delta Force", "Rainbow" та її варіації, "Hitman" і багато інших. У них ворогами є люди, і завдання гравця, керуючи героєм, чи, найчастіше, командою, перемогти всіх "поганих хлопців".

Комп'ютерна гра такого типу може формувати певні корисні навички. Гравець навчається швидким, точним і тонким рухам рук. Інша навичка, яку формують "Doom"-подібні ігри, це здатність працювати у ритмі, заданому особливостями процесу. З цього погляду "Doom"-подібна гра є засобом навчання швидкому опрацюванню інформації в умовах високої нервової напруги та дефіциту часу.

Ігри з елементами рольового моделювання дуже різноманітні. Більшість з них мають свій складний і захоплюючий світ. Наприклад, "Morrowind" (1, 2, 3) має свою історію, свої раси розумних істот, що дає можливість гравцеві прожити життя одного з персонажів з усіма його особливостями. Частина рольових ігор дуже схожа на звичайний "action", інші набагато ближчі до стратегій. Наприклад, серія "Heroes of Might and Magic" (2, 3, 4), що захоплює вже третє покоління гравців, за багатьма параметрами близька до стратегій, а подібна до неї "Might and Magic" (6,

8, 9), як рольовий "action", перебуває на протилежному полюсі. "Aliens vs Predator" (1 і 2) містить у собі як рольові елементи, так і всі основні елементи "action".

Особливістю таких ігор є можливість обирати героя чи групу героїв, розвивати в них специфічні здібності, а іноді, як у "Aliens vs Predator", навіть бачити навколишній світ по-різному – залежно від героя, від імені якого ведеться гра. Цікаво, що в цьому випадку підлітки і діти набагато швидше адаптуються до зміни сприйняття. Багато дорослих людей, що спробували грати "Чужого" з його незвичайним зором, відчувають запаморочення і нудоту. Зате не менш незвичайний "потрійний" зір "Хижака" легко сприймається всіма. Скоріш за все, це пов’язано з особливостями програми, що описує рух цих істот.

Ігри подібного типу розвивають гнучкість мислення, здатність ставитися до будь-яких героїв та істот як до одного з варіантів різноманітного світу. Правота і неправота в цих іграх, як і в реальному світі, відносні. Усе залежить від того, на чиєму ти боці. Та навіть при відносності "правди" більшість з ігор цього типу має чітке протиставлення "добра" і "зла".

Для більшості подібних ігор не досить тільки швидкої реакції. Набагато частіше необхідне розвинене комбінаторне мислення та здатність прогнозувати неоднозначні ігрові ситуації. Кожна з ігор цього розділу формує внутрішню переконаність у тому, що немає непереможних монстрів. Монстри небезпечні, та герой може знайти засіб для перемоги. А для перемоги, на відміну від реального світу, є можливість зробити кілька спроб. Наявність функцій "autosave" та "save" значно розширює простір пошуку кращого рішення у кожній ситуації.

Стратегії. Стратегічні ігри неоднорідні: одні пропонують тільки "мир", інші – "війну", треті – те і те одночасно. У мирному жанрі до лідерства близьке сімейство ігор "SimCity": "SimCity-2000", "Sim Tower", "SimCity-3000". Для військового жанру найтиповіша гра "Warhammer: Dark Omen". Війна і мир, як жанр стратегічних ігор, найбільший за кількістю назв. Багато гравців вважають ігри "Age of Empire" і "Civilization", а також їх похідні еталоном сучасних стратегічних ігор.

Найближчим прототипом подібних стратегічних ігор є шахи. Ілюзія шахів, що ожили, реалізувалася у вигляді стратегічної комп’ютерної гри: людина дивиться

на екран монітора і бачить у тривимірному просторі макет місцевості з живими істотами. Віртуальну країну можна наблизити, віддалити, оглянути під іншим кутом зору. Її розмір може сягати сотень квадратних кілометрів. На ній існують і переміщуються з волі гравця і за власним алгоритмом безліч фігур різних видів. Їх зовнішній вигляд постійно змінюється. Військові загони зазнають втрат, але здобувають бойовий досвід, цивільні об'єкти нарощують чи втрачають міць. Часовий проміжок гри може досягати тисячоріч.

Як правило, ігрове поле є макетом місцевості. Іноді воно розмічене на клітинки, як у "SimCity", частіше – ні, але наявність структури визначає все, що відбувається у більшості ігор. Фігура – це об'єкт, яким гравець може керувати як цілим. Частина фігур статичні: наприклад, заводи, електростанції, будинки. Ступінь і швидкість їх розвитку залежать від усіх інших фігур дошки. Інші фігури рухливі, наприклад, смерч, тарілки, що літають, чудовиська. У військових іграх більшість фігур рухливі. Спочатку гравець розставляє свої фігури – загони кінноти, піхоти, лучників – на відведеному для цього місці, а потім може їх відправити в будь-яку зону дошки. Ходи супротивника розраховує комп'ютер.

Стратегічна гра складається з ряду рівнів. У мирних іграх основний показник якості гри – чисельність населення. Коли він перевищує заздалегідь задане для даного рівня число, у гравця з'являється можливість використовувати додаткові види фігур. У військових іграх головне – розгром ворога. Якщо гравцеві це вдається, то він одержує нову дошку з іншим рельєфом місцевості та потужніші фігури. Ігри, що поєднують війну і мир, вимагають спочатку побудувати інфраструктуру для армії, а вже потім знищувати ворога. Військова мета у цьому випадку тісно пов'язана з мирною.

Більшість ігор не дозволяє гравцеві бачити всю дошку відразу. Гра звичайно відбувається на дошці, більшій поля зору. Вірніше, дошку можна вмістити, але для цього доведеться її так зменшити, що стане неможливо розрізнити, що на ній відбувається. Велику частину ігрового часу гравець розглядає дошку у великому масштабі. Та щоб виграти, йому необхідно точно уявляти, що і де на дошці

відбувається. Для цього гравець повинен мати внутрішній образ дошки. З ним він співвідносить усі свої дії, постійно уточнюючи і корегуючи його в міру забування.

У процесі гри формуються навички системного аналізу. Гравцеві необхідно швидко і правильно опрацьовувати інформацію, а потім на її основі прогнозувати розвиток подій. Ще одна навичка, яку формують ігри подібного типу, це досвід роботи з "чорною скринькою" як пристроєм з невідомим принципом роботи, що має вхід і вихід.

Експериментуючи з введенням даних, гравцю необхідно домогтися потрібної реакції на виході. На початку гри майже усі фігури для гравця – "чорна скринька". Якусь інформацію про їх взаємозв’язки дають опис та система підказок. В основному гравець робить припущення, керуючись здоровим глуздом. Інколи вони виправдовуються, інколи – ні. В іграх цього типу можливе формування навичок експериментування та пошуку оптимальних варіантів діяльності в умовах дефіциту інформації.

Квест. Слово "quest" означає пошук, предмет, що відшукується, пошуки пригод, дізнання. У цих іграх реалізується одне з занять, що захоплюють людей різного віку – розгадування загадок. Людську потребу розкривати таємниці давно використовують театр, кінематограф і література. Мистецтво та засоби масової інформації могли запропонувати тільки пасивну участь у таких іграх. Гравець у комп’ютерній грі, на відміну від читача чи глядача, розгадує загадку сам.

У віртуальному світі квесту воля гравця у пересуванні набагато менша, ніж у "Doom" -подібній грі. У визначеній зоні простору гравець обирає шлях, потім пасивно спостерігає переміщення до нового місця. За структурою квест схожий на більшість комп’ютерних довідкових та інформаційних систем з гіпертекстом. Можливість перейти зі сторінки на сторінку в квесті залежить від попередніх дій гравця: поки він не виконає їх у мінімальному обсязі, це неможливо. У момент переходу гра демонструє мультфільм чи фрагмент відеофільму. На кожен свій крок гравець одержує реакцію. Квест вчить враховувати зворотний зв’язок та визначати, вірні припущення гравця чи ні. В іграх цього типу можливе формування навичок дедуктивного та індуктивного мислення.

Ігри, що імітують. Найчастіше комп’ютерні ігри цієї категорії імітують керування транспортним засобом. Вони дають можливість "приміряти" нову соціальну роль: пілота літака, гелікоптера, командира танка, водія реальних і фантастичних машин. Ігри, що імітують, більше за інші типи ігор використовують історичні факти, особливо з розвитку техніки. Ігри, що імітують, широко використовуються не тільки для гри, але й для формування навичок керування реальними об’єктами і процесами. Завдяки цьому в іграх такого типу можливе формування навичок, необхідних при реальному управлінні технікою.

Віртуальне казино. Розроблено безліч ігор, що імітують реальні азартні ігри. Кожна з них будується на дотриманні тих же правил, що й у реальній грі. Тому основні прийоми цілком збігаються з грою в реальному світі. Основна різниця в тому, що не одержуєш реальних грошей при виграші. Хоча при використанні Інтернету ці ігри перетворюються на звичайні азартні. Сформовані навички не відрізняються від тих, що формуються у звичайних іграх цього типу. Для успішної гри в карти необхідна логіка, схильність до дедуктивного та індуктивного мислення, навички до рефлексії. В іграх з автоматами, де є вірогідність виграшу залежно від дій гравця чи результату якихось подій, наприклад, перегонів, може формуватися навичка врахування у реальному житті положень теорії вірогідності.

Позитивний потенціал більшості комп’ютерних ігор реалізується далеко не завжди. І це залежить здебільш не від самої гри, а від людини, що грає, від того, який мотив переважає при включенні до гри. Крім основного мотиву розваги, гра може реалізувати інші мотиви. У залежності від мотивів другого плану можуть формуватися абсолютно різні для різних гравців навички та уміння. Реалізація мотиву тренінгу призводить до формування навичок у сфері, що тренується, а мотив компенсації внутрішніх проблем скоріш матиме результатом формування механізмів психологічного захисту. У зв’язку з розмаїттям мотивів гра може сприяти як підготовці до зустрічі з реальністю, так і втечі від неї.

У міру того, як захоплення комп’ютерними іграми стає все поширенішим, проблема їх впливу на психіку людини та формування різних навичок потребує більшої уваги.

Для адекватного прогнозування соціальних та індивідуальних наслідків комп’ютеризації населення необхідне формування національної і міжнародної програм дослідження цього феномена.

2 РОЗРОБКА СТРУКТУРИ ДОДАТКУ ТА ТЕХНОЛОГІЇ ОПТИМІЗАЦІЇ

2.1 Моделювання проекту

2.1.1 Аналіз аналогів

Провівши аналіз найближчих аналогів було виявлено наступні переваги та недоліки застосунку, які зображені на Рисунок 2.1 – Візуальне представлення класів:

Категорії порівняння	**The Banner Saga**	**Ach Of Gods**	**The Art of Tahira**	**The Loss Of Bagrash**
Сюжет	+	+	+	-
Можливість гри разом на одному пристрої	-	-	-	+
Підтримка української мови	-	+	-	+
Можливість зіграти самому	+	+	+	+

Рисунок 2.1 – Візуальне представлення класів

2.1.2 Візуальне представлення класів

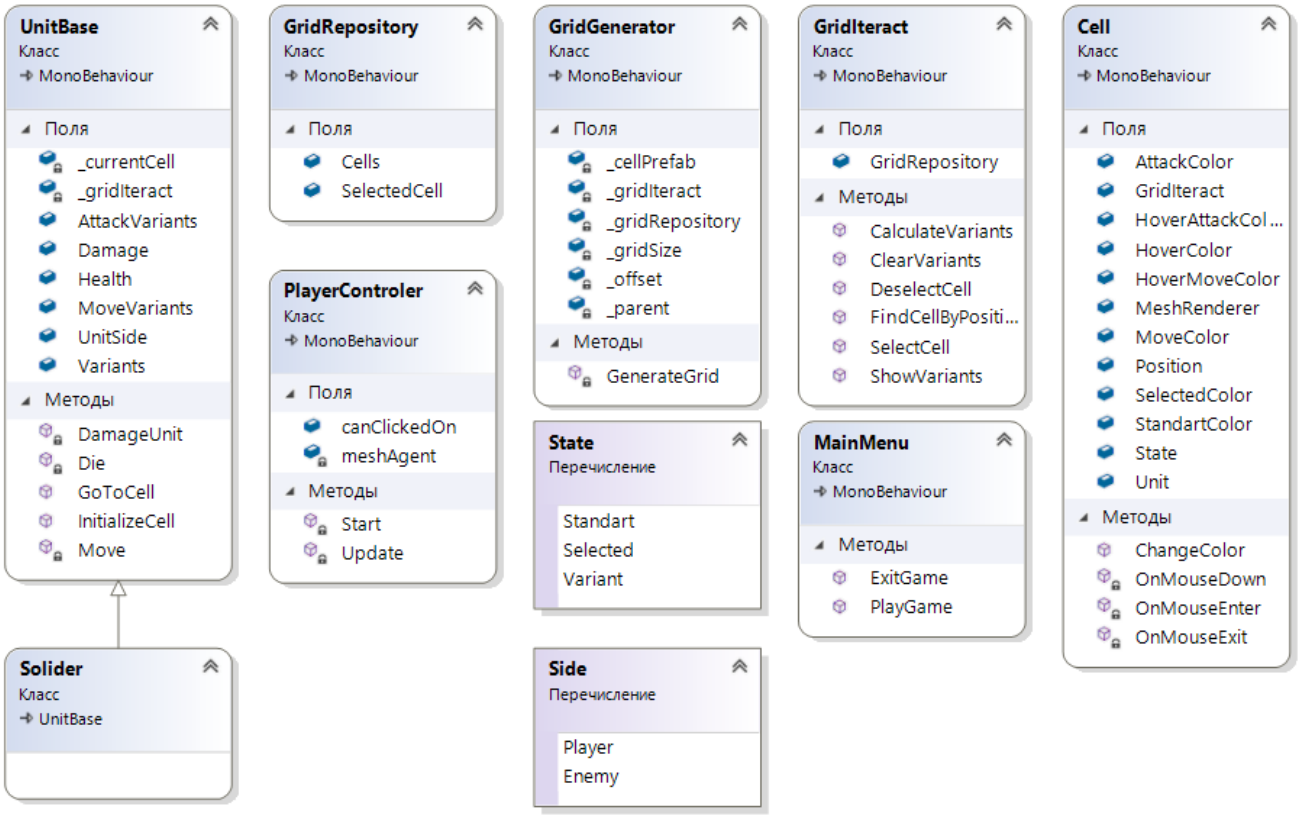


Рисунок 2.2 – Візуальне представлення класів

2.1.3 Діаграма прецедентів додатку

При створенні програмного застосунку він повинен направляти користувача аби той зміг виконати ті дії які пропонує застосунок таким чином щоб для користувача це було якомога простіше і ясніше. При створенні програмного додатку потрібно поставити вимоги які він має реалізувати. Не можливо визначитися з вимогами на основі одного користувача так як різним користувачам одна реалізація може бути зручною для досягнення цілі а іншому навпаки здаватися занадто важкою і не зрозумілою.

Для більш детального розуміння як саме повинен працювати програмний модуль та що від нього потребується використовують опис через варіанти використання такі як: Use Case діаграми або діаграми прецедентів.

Наведені вище варіанти використання(діаграми Use Case, прецедентів) потрібні зазвичай для визначення функціональних вимог програмного застосунку і

напрямляють увесь процес розробки. Такі дії як: аналіз, тестування, проектування виконуються на основі вище вказаних варіантів використання. Щоб зрозуміти як результат дій яку хоче досягнути користувач і як це буде впливати на програмний застосунок та як на це повинні реагувати компоненти цього застосунку для досягнення цілі поставленої користувачем застосунку, проводиться аналіз та проектування такі дії стають можливими завдяки варіантам використання.

Завдяки варіантам використання процес тестування стає простіше оцінити реалізацію вимог користувачів і точніше також з’являється можливість провести покрокову перевірку поставлених вимог.

Після того як вимоги були визначені за допомогою варіантів використання можливим для розгляду стає питання про “що користувачі застосунку очікують від нього”. Завдяки такому підходу відкривається можливість для пошуку саме тих функцій які по-справжньому необхідні для користувача та навпаки відсіювання функцій які будуть непотрібні це також зменшить час та кошти на розробку застосунку.

Про діаграму варіантів використання на цій діаграмі графічно зображується так званий актор для якого буде зображено дію застосунку яку він може відтворити і ті самі дії Use Case які хоче відтворити користувач він же актор. Актор діаграми позначається чоловічком а горизонтальним овалом з описом самої дії в середині.

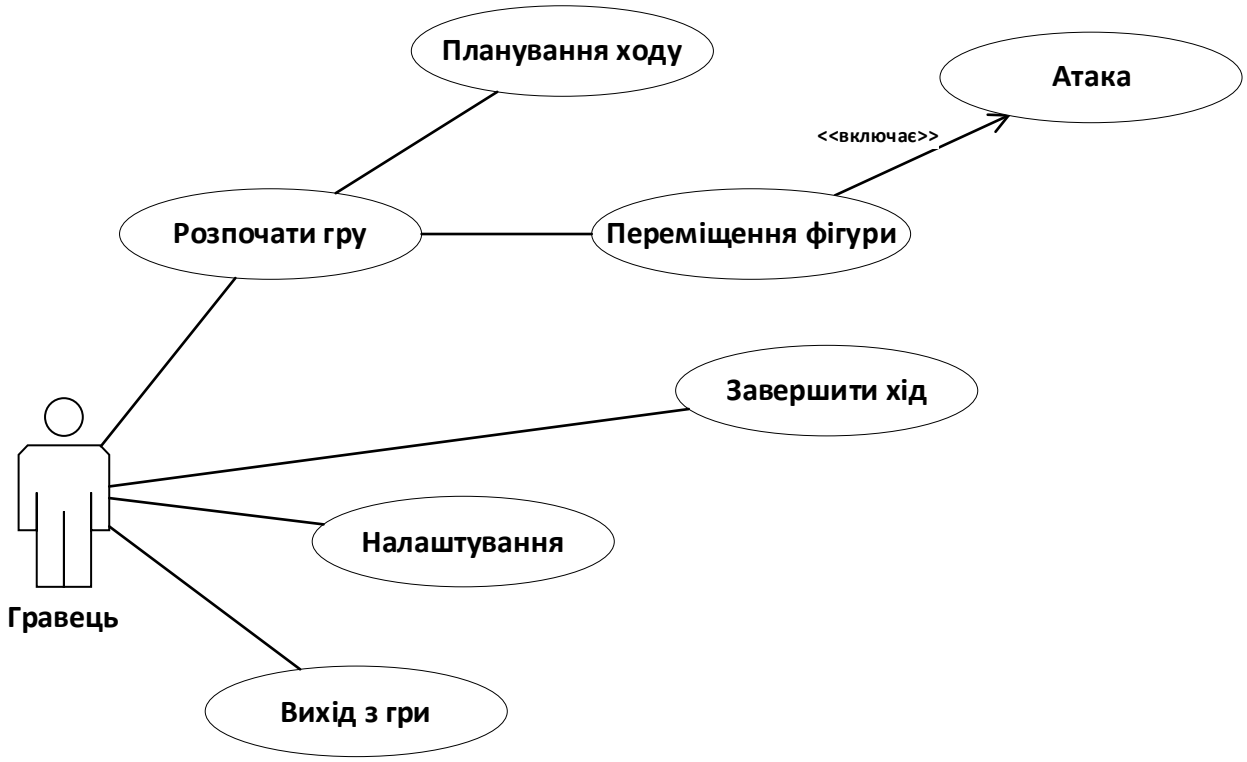


Рисунок 2.3 – UML діаграма прецедентів застосунку

Користувачами застосунку буду виступати гравці які бажають відволіктися від своїх справ, відпочити та розважитися.

Діаграма класів (Class Diagram) – діаграма мови UML, на якій представлена сукупність декларативних або статичних елементів моделі, таких як класи з атрибутами і операціями, а також зв'язують їх відносини.

Діаграма класів дозволяє формалізувати логічну модель вашої програми, на рівні структурних елементів системи, тобто класів. Вона надає статичне представлення про структуру програми – з яких класів вона складається, які зв’язки між цими класами, з чого складається кожний клас.

Модель кожного класу є шаблоном майбутнього об’єкту, який створюється на основі цього класу. Кожна програма може містити чисельну кількість різних об’єктів. Як відомо, об’єкти мають стан та поведінку, відповідно до концепції ООП. На рівні класу стан описується полями даних, а поведінка – методами.

2.2 Структура застосунку

2.2.1 Структура проекту Unity

По мірі росту проекту збільшується кількість різних файлів в ньому з часом кількість файлів буде збільшуватися і в один момент прийде розуміння що в цій купі стає важко знайти щось конкретне і потрібно відсортувати все це і створити структуру для зручної і комфортної роботи над проектом.

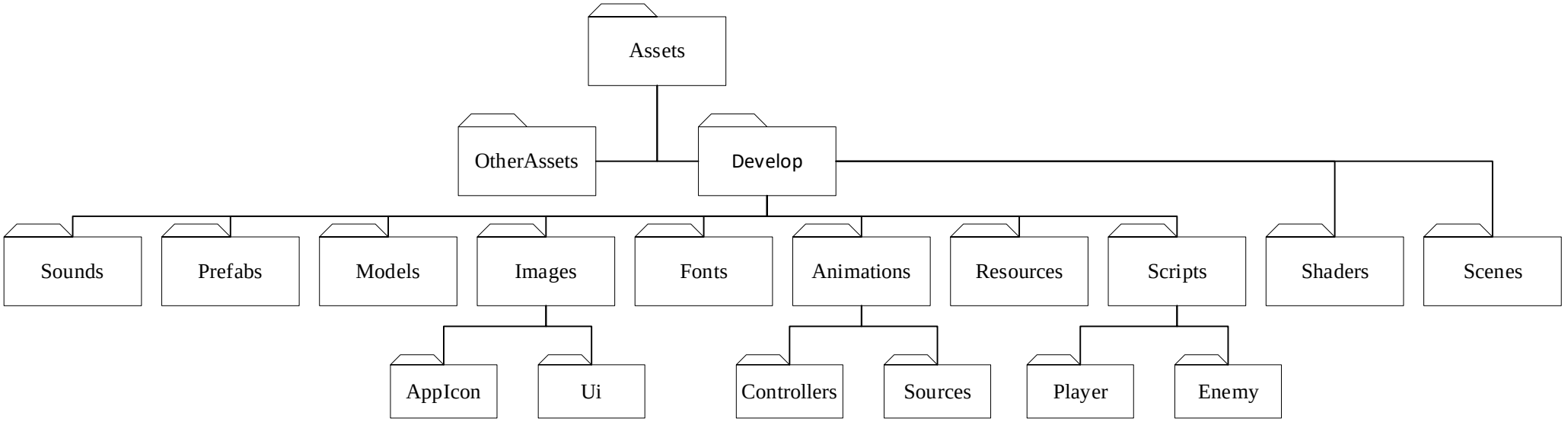


Рисунок 2.4 - Структура застосунку

- Assets - Кореневий розділ проекту Unity всі ассети повинні бути в цьому розділі.
    - Develop – знаходяться всі ассети, специфічні для цього проекту.
        - Animations
            - Controllers - знаходяться графи AnimationController, маски для аніматору.
            - Sources - знаходяться анімації в форматі FBX / DAE.
        - Fonts - знаходяться шрифти.
        - Images – знаходяться 2D – зображення які не відносяться до 3D моделей.
            - AppIcon - знаходяться піктограми застосунку.
            - UI - знаходяться зображення, зв’язані з ігровим UI.
                - Atlases - знаходяться атласи Ui.
                - Sources - знаходяться вихідні зображення для складання атласів. Кожен атлас - окрема папка з тим самим ім'ям, що й атлас.
        - Models - знаходяться 3D моделі.
        - Prefabs - знаходяться префаби.
        - Resources - знаходяться ресурси для динамічного завантаження.
        - Scenes - знаходяться сцени.
        - Scripts - знаходяться скрипти.
            - Player - знаходяться скрипти пов’язані з персонажем гравця.

- Enemy - знаходяться скрипти пов’язані з ворогами.

- Shaders - знаходяться шейдери.

- Sounds - знаходяться музика та звукові ефекти.

- OtherAssets –стороній ассет з Asset Store кожний сторонній ассет зберігається в окремому розділі.

З вище описаної структури видно що усі дані проекту знаходяться в розділі під назвою Develop, це зроблено з метою відокремити сторонні ассети імпортовані з Asset Store задля того щоб було легше їх оновлювати до актуальної версії.

2.3 Технології оптимізації

При розробці в Unity початківцю чи тим хто розробляє не великі застосунки можуть і не задумуватися про таку річ як оптимізація застосунку, в Unity порівняно з іншими конкурентами доволі низький поріг входження це і добре і викликає деякі проблеми. Можна знайти вже готові ассети та написати пару простих скриптів розробивши досить простий застосунок але швидше за все в майбутньому коли буде розширюватися застосунок виявиться що без оптимізації він стане не придатним до використання і прийдеться можливо навіть все переробляти тому не варто забувати про оптимізацію навіть в невеликих застосунок бо коли такий застосунок навантажує пристрій не гірше середнього по величині застосунку це не добре.

2.3.1 Unity Profiler

Unity Profiler – це інструмент який допомагає отримувати інформацію про продуктивність роботи вашого застосунку. Отримувати інформацію про продуктивність застосунку можна і з інших пристроїв які знаходяться під’єднані до ПК або знаходяться в одній локальні мережі з ПК таким чином можна проводити

тестування продуктивності і на різних платформах. Також можна ту саму інформацію отримати і з ПК в редакторі Unity.

Profiler надає можливість відображати та збирати інформацію про продуктивність застосунку в різних сферах системи таких як: ЦП, пам’яті, засіб віртуалізації та звук. Завдяки цьому інструменту можна відстежити інформацію про навантаження застосунку на ту чи іншу сферу. Можна точно відстежити як код, налаштування сцени, ассети, рендер камери, налаштування складення рішення впливають на продуктивність застосунку. Відображаються статистка у вигляді діаграм на якій будуть відображатися перепади швидкодії застосунку.

На додачу можна також використати низькорівневий вбудований API-інтерфейс Profiler для розширення профілювання швидкодії вбудованого плагіна коду або для підготовки та відправлення даних профілювання.

2.3.2 Frame Debugger

Frame Debugger надає можливість призупинити процес гри під час тестування у середовищі розробки Unity так можна відстежувати виклики рендеру в конкретний момент також після призупинення можна відтворювати наступні кадри один за одним.

Є список який показує послідовність викликів drawcall виглядає він як ієрархія яка показує звідки воно родом. Також є панель розміщена праворуч від списку в ній відображаються додаткова інформація та деталі, такі як: геометрії об’єктів та шейдер який використовується для рендеру.

Клацнувши по елементу зі списку буде відображено сцену у тому вигляді, в якому вона з’явилась до цього виклику. На панелі інструментів є кнопки навігації для переміщення на крок вперед або назад можна використовувати клавіші стрілок на клавіатурі. Також слайдер у верхній частині вікна дозволяє швидко переміщуватися по списку викликів аби швидше можна було знайти шукомий елемент. Якщо виклик задовольняє геометрію GameObject ще об’єкт буде відображено в панелі ієрархії для спрощення його ідентифікації.

На вкладці Game буде відображено вміст RenderTexture якщо відбувається його рендер з обраним викликом малюванням.

2.3.3 Object pool

Object pool — відмінний спосіб оптимізувати ваші проекти та знизити навантаження на ЦП за необхідності швидкого створення та знищення ігрових об'єктів. Це хороша практика і шаблон проектування, про які слід пам'ятати, щоб допомогти розвантажити обчислювальну потужність ЦП для виконання важливіших завдань і не перевантажуватися викликами створення і знищення, що повторюються. Це особливо корисно при роботі з кулями у шутері з видом зверху.

Object pool — це творчий шаблон проектування, який заздалегідь створює екземпляри всіх об'єктів, які вам знадобляться в будь-який момент перед грою. Це усуває необхідність створення нових об'єктів або знищення старих під час гри. Пули об'єктів в основному використовуються для підвищення продуктивності: в деяких випадках пули об'єктів значно підвищують продуктивність, коли проект багаторазово створює і знищує той самий ігровий об'єкт у швидкій послідовності. Він працює, створюючи певну кількість ігрових об'єктів перед запуском гри і просто деактивує або активує необхідні ігрові об'єкти, фактично просто переробляючи ігровий об'єкт і ніколи не знищуючи його.

Object pool — важлива концепція для розуміння через природу деяких ігрових об'єктів і того, як часто вони будуть створюватися або знищуватись під час гри. Коли ви обробляєте безліч викликів інстанцування та знищення одного ігрового об'єкта, можливо, настав час подумати про реалізацію пулу об'єктів.

Проблема

Пули об'єктів (інакше звані пулами ресурсів) використовуються керувати кешуванням об'єктів. Клієнт, який має доступ до пулу об'єктів, може уникнути створення нового об'єкта, просто звернувшись до пулу за вже створеним об'єктом. Зазвичай пул зростає, тобто. пул сам буде створювати нові об'єкти, якщо пул порожній, або ми можемо мати пул, який обмежує кількість об'єктів, що створюються.

Бажано тримати всі об'єкти Reusable, які зараз не використовуються, в одному пулі об'єктів, щоб ними можна було керувати за допомогою однієї узгодженої політики. Досягнення цієї мети клас Reusable Pool спроектований як синглтон.

Структура

Загальна ідея патерну Connection Pool полягає в тому, що якщо екземпляри класу можуть бути використані повторно, ви уникаєте створення екземплярів класу, використовуючи їх повторно.

Багаторазові - екземпляри класів у цій ролі співпрацюють з іншими об'єктами протягом обмеженого часу, потім вони більше не потрібні для цього співробітництва.

Client - екземпляри класів у цій ролі використовують об'єкти Reusable.

ReusablePool - екземпляри класів у цій ролі управляють об'єктами Reusable для використання об'єктами Client.

Зазвичай бажано тримати всі об'єкти Reusable, які зараз не використовуються, в одному пулі об'єктів, щоб ними можна було керувати за допомогою однієї узгодженої політики. Для досягнення цієї мети клас ReusablePool спроектований як клас-одинак. Його конструктор(и) є приватним, що змушує інші класи викликати його метод getInstance для отримання єдиного екземпляра класу ReusablePool.

Об'єкт Client викликає метод acquireReusable об'єкта ReusablePool, коли йому потрібен об'єкт Reusable. Об'єкт ReusablePool підтримує колекцію об'єктів Reusable. Він використовує колекцію об'єктів Reusable для утримання пулу об'єктів Reusable, які не використовуються.

Якщо на момент виклику методу acquireReusable в пулі є об'єкти Reusable, він видаляє об'єкт Reusable з пулу і повертає його. Якщо пул порожній, метод acquireReusable створює об'єкт Reusable, якщо може. Якщо метод acquireReusable не може створити новий об'єкт Reusable, він чекає, поки об'єкт Reusable не буде повернений до колекції.

Клієнтські об'єкти передають об'єкт Reusable методом releaseReusable об'єкта ReusablePool, коли закінчують роботу з об'єктом. Метод releaseReusable повертає об'єкт Reusable в пул об'єктів Reusable, які не використовуються.

У багатьох програмах патерну Object Pool є причини обмеження загальної кількості об'єктів Reusable, які можуть існувати. У таких випадках об'єкт ReusablePool, який створює об'єкти Reusable, відповідає за те, щоб не створювати більше заданої максимальної кількості об'єктів Reusable. Якщо об'єкти ReusablePool відповідають за обмеження кількості об'єктів, які вони створюють, то клас ReusablePool матиме метод для вказівки максимальної кількості створюваних об'єктів. На наведеній діаграмі цей метод позначений як setMaxPoolSize.

Висновок. Object pool чудовий шаблон проектування для оптимізації проекту та зниження навантаження на центральний процесор пристрою у випадку коли потрібно швидко створювати чи видаляти ігрові об’єкти.

2.4 Технології рендерингу

2.4.1 Shader Graph

Shader Graph дозволяє візуально створювати шейдери та бачити результати в реальному часі. Ця система на основі вузлів відкриває поле діяльності для художників та інших членів команди – просто з'єднайте вузли у граф-мережу.

Shader Graph забезпечує швидкий час ітерацій на кожному етапі створення шейдера. Майже кожен вузол має вбудований перегляд, що дозволяє бачити покроковий результат. Сам граф має загальний попередній перегляд, тому ви можете бачити кінцеві результати роботи шейдера. Якщо шейдер застосовується до моделі в сцені, шейдер оновлюється миттєво та зберігається, що дозволяє отримувати оновлення за секунди. Всі незручності, пов'язані з синтаксисом коду, компіляцією та створенням, усунуті, тому немає жодних бар'єрів між вами та вашим творчим баченням.

2.4.2 Shader Graph

Houdini Engine підтримує глибоку інтеграцію Houdini та його процедурного робочого процесу в рамках конвеєра Studio. Houdini Engine функціонує як плагін для інших додатків та як неграфічне рішення для пакетної обробки та розподілу важливих завдань.

Houdini Engine привносить процедурний підхід на основі вузлів у улюблений додаток. Створені мережі, які визначають рецепт, який можна застосовувати знову і знову, а потім можна обернути їх для створення смарт-активів. Можна поділіться цими активами з колегами, які зможуть завантажити їх безпосередньо в 3D-програми створення моделей або в ігрові редактори.

Houdini Engine також можна використовувати для пакетної обробки Houdini у неграфічному режимі. Ви можете використовувати ліцензії Houdini Engine у пакетному режимі для розподілу широкого спектру важливих завдань, таких як рендеринг, динамічне моделювання, рух та хешування геометрії.

2.4.3 Light Baking

Запікання світла це створення текстурних карт які базуються на основі освітлення сцени. Щоб використовувати Light Baking необхідний Ultra Render. Ралістичне освітлення Ultra Render з трасуванням шляхів це основа створення карт освітлення.

Створені карти освітлення можна застосовувати на карти матеріалів та текстури. Ефекти освітлення по типу: тіней та зміни кольору на поверхні об’єкта такі ефекти проектувати будуть карти освітлення.

Таким чином можна налаштувати сцену з цікавими підходами освітлення або просто покращити те що не влаштовувало. Це зручний спосіб який також дозволяє зменшити кількість джерел освітлення та підвищити продуктивність застосунку.

2.4.4 LOD

Алгоритми динамічного рівня деталізації геометрії (LOD) - це дуже популярні та потужні алгоритми, які забезпечують високий рівень оптимізації

продуктивності рендерингу при збереженні деталізації за рахунок використання менш детальної геометрії для об'єктів, які знаходяться далеко, занадто малі або інакше менш значущі для якості кінцевого рендерингу. Багато які з них використовується з самого початку розвитку технологій комп'ютерної графіки в тій чи іншій формі присутні в сучасних програмах CAD, відеоіграх та інших графічних додатках. Якщо раніше визначення відповідної LOD геометрії було завданням центрального процесора, то із сучасним обладнанням це можна перекласти на GPU, який чудово справляється з паралельною обробкою великої кількості об'єктів.

2.4.5 Texture atlas

Текстурний атлас об'єднує кілька зображень на одну текстуру. На деяких платформах текстури OpenGL повинні мати розмір, що дорівнює ступеню 2. Завантаження зображень окремо вимагало б багато прокладок, що означає нераціональне використання пам'яті GPU. Крім того, наявність всіх зображень в одній текстурі дозволяє використовувати пакетну обробку викликів малювання: іншими словами, кілька спрайтів можуть бути намальовані одним викликом.

Хоча інструмент був розроблений для iPhone, він корисний для більшості платформ, включаючи Android та ПК. Інструмент написаний на Java та працює під Windows, Mac та Linux. Код написаний мовою C++ та розроблений для переносимості.

2.4.6 DOTS

DOTS підвищує продуктивність ігор на багатоядерних процесорах, не вимагаючи розробки складних алгоритмів для паралельних обчислень. Незважаючи на те, що більшість пакетів Unity, що належать до DOTS, в даний час перебувають у статусі попередньої версії, деякі з них можуть зробити проект у критичних для продуктивності аспектах значно кращими.

DOTS - це зручна пісочниця для розробки безпечного багатопотокового коду, що підвищує продуктивність, оптимізує тепловиділення та енергоспоживання мобільних пристроїв гравців. Крім того, перехід від об'єктно-орієнтованого до

інформаційно-орієнтованого підходу спрощує багаторазове використання коду, та дозволяє легко зрозуміти і доповнити його при необхідності.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

Застосунок представляє собою гру в жанрі «покрокова стратегія», на розробку в такому жанрі надихнула гра «The Banner Saga». Також сподобалася реалізація бойової системи в ній. Наведена гра представляє собою гру для одного гравця, тому як аналог гри було вирішено розробити схожу, але з можливістю гри вдвох на одному ПК.

Даний жанр ігор представляє собою поле на якому фігури чи юніти мають можливість пересуватись та атакувати ворожих юнітів на полі під час свого ходу.

Перш за все розробка гри починається з реалізації ігрового поля бою, поле бою представлене, як двовимірний масив кліток на якому розміщуються юніти.

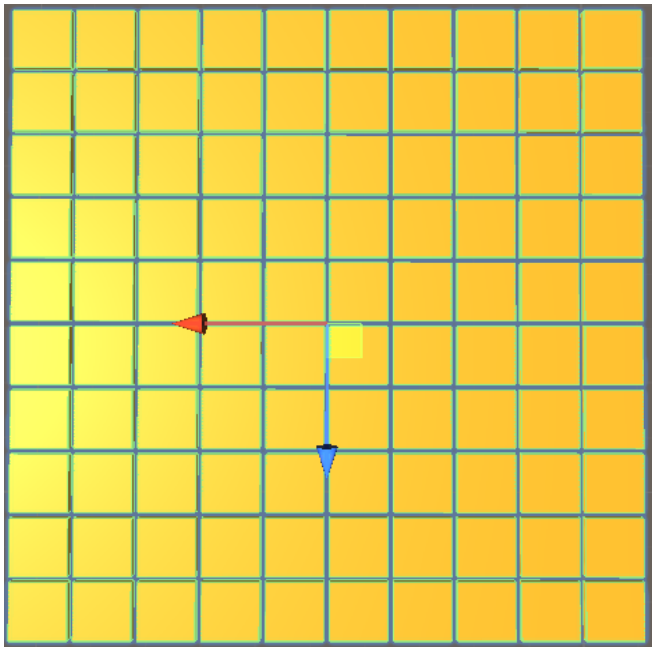


Рисунок 3.1 – Представлення ігрового поля

Розмір поля залежить від вказаної змінної, для генерації кожної комірки використовується префаб клітинки поля, встановлюється розмір поля в залежності від кількості клітинок в ньому.

За допомогою циклів поле заповнюється клітинками, встановлюються проміжки між ними та кожній клітинці присвоюється власні координати X та У.

```
[SerializeField] private Vector2Int _gridSize; /*довжина сітки по x,y*/
[SerializeField] private Cell _cellPrefab; /*префаб клітинки*/
[SerializeField] private float _offset; /*відступ між клітинками*/
[SerializeField] private Transform _parent; /*обєкт на якому зберігаються обєкти*/
[SerializeField] private GridRepository _gridRepository;
[SerializeField] private GridIteract _gridIteract;

[ContextMenu("Generate grid")]
Ссылок: 0
private void GenerateGrid() /*основна логіка генерації сітки*/
{
    _gridRepository.Cells.Clear();
    _cellPrefab.GridIteract = _gridIteract;

    var cellSize = _cellPrefab.GetComponent<MeshRenderer>().bounds.size; /*отримуємо розмір сітки*/

    for (int x = 0; x < _gridSize.x; x++) /*генерація самої сітки*/
    {
        for (int y = 0; y < _gridSize.y; y++)
        {
            var position = new Vector3(x * (cellSize.x + _offset), 0, y * (cellSize.z + _offset)); /*позиція клітинки з урахуванням відступів*/

            var cell = Instantiate(_cellPrefab, position, Quaternion.identity, _parent); //обєкт клітинки

            cell.name = $"X: {x} Y: {y}";
            cell.Position = new Vector2(x, y);

            _gridRepository.Cells.Add(cell);
        }
    }
}
```

Рисунок 3.2 – Метод генерації ігрового поля

Наступними описуються методи взаємодії з клітинкою. Клітинка має свій стан, в залежності від якого може змінюватись колір, так при наведенні або при натисканні клітинка виділяється.

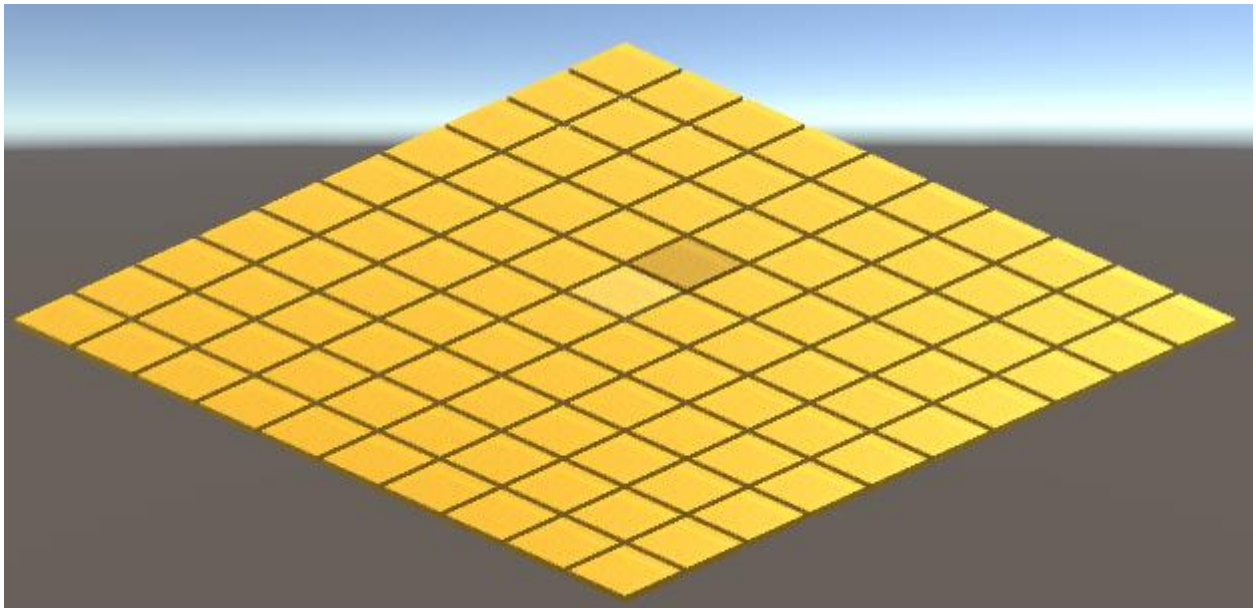


Рисунок 3.3 – Приклад виділеної клітинки

```
public Color StandartColor; /*стандартний колір клітки*/
public Color HoverColor; //колір клітинки при наведенні
public Color HoverMoveColor; /*колір клітинки коли по ній можна пройти*/
public Color HoverAttackColor; /*колір клітинки коли на ній ворог в діапазоні атаки*/

public Color SelectedColor;
public Color MoveColor;
public Color AttackColor;

public MeshRenderer MeshRenderer;

public UnitBase Unit; /*юніт на клітинці*/
[HideInInspector] public Vector2 Position; /*для зміни позиції клітинки*/
[HideInInspector] public State State; /*для збереження стану клітинки*/

public GridIteract GridIteract;

Ссылок: 11
public void ChangeColor(Color color) //метод для зміни кольору
{
    MeshRenderer.material.color = color;
}

Сообщение Unity | Ссылок: 0
private void OnMouseEnter()
{
    if (State == State.Standart)
    {
        ChangeColor(HoverColor); /*зміна кольору клітинки принаведенні миші*/
    }
    else if (State == State.Variant)
    {
        if (Unit == null)
        {
            ChangeColor(HoverMoveColor);
        }
        else
        {
            ChangeColor(HoverAttackColor);
        }
    }
}
```

```
private void OnMouseDown() /*при натисканні на клітинку*/
{
    if (State == State.Selected)
    {
        GridIteract.DeselectCell(this); //прибрати виділення
    }
    else if (State == State.Variant)
    {
        GridIteract.GridRepository.SelectedCell.Unit.GoToCell(this);
    }
    else if (State == State.Standart)
    {
        if (GridIteract.GridRepository.SelectedCell != null)
        {
            GridIteract.DeselectCell(GridIteract.GridRepository.SelectedCell);
        } //виділяти клітинку

        GridIteract.SelectCell(this);
    }
}

Сообщение Unity | Ссылок: 0
private void OnMouseExit()
{
    if (State == State.Standart)
    {
        ChangeColor(StandartColor); /*повернення стандартного кольору коли клітинка не виділена*/
    }
    else if (State == State.Variant)
    {
        if (Unit == null)
        {
            ChangeColor(MoveColor);
        }
        else
        {
            ChangeColor(AttackColor);
        }
    }
}
```

Рисунок 3.4 – Методи взаємодії з клітинкою

Якщо було обрано клітинку в якій знаходиться розміщений юніт, з’являються варіанту його ходу, також якщо ворожий юніт знаходиться в діапазоні переміщення, то клітинка в якій стоїть ворожий юніт підсвічується червоним.

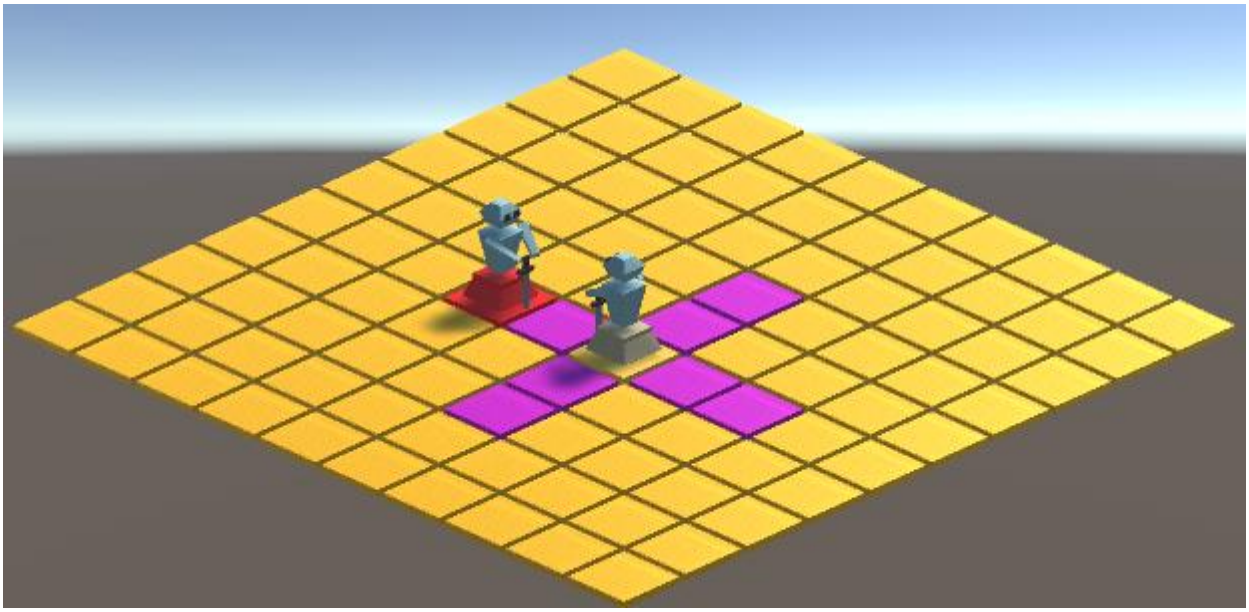


Рисунок 3.5 – Виділення юніту та варіанти ходу

При виділенні юніту прогнозуються варіанти його ходу, тобто з точки в якій розміщений юніт передається список варіантів для ходу, що містить певний виділений юніт.

```
public void ShowVariants(List<Cell> variants) /*Відображення варіатів ходу, аргумент список з збереженими варіантами ходу*/
{
    foreach (Cell variant in variants)
    {
        variant.State = State.Variant;

        if (variant.Unit == null)
        {
            variant.ChangeColor(variant.MoveColor);
        }
        else
        {
            variant.ChangeColor(variant.AttackColor);
        }
    }
}

Ссылок: 2
public void ClearVariants(List<Cell> variants) /*аргумент варіантс з поточними варіантами ходу*/
{
    foreach (var variant in variants)
    {
        variant.State = State.Standart;
        variant.ChangeColor(variant.StandartColor);
    }
}
```

Рисунок 3.6– Методи відображення варіантів ходу юніта

```
public List<Cell> CalculateVariants(Cell unitcell, List<Vector2> movevariants, List<Vector2> attackvariants)
{
    var variants = new List<Cell>();
    Cell cell;

    foreach (Vector2 offset in movevariants)
    {
        cell = FindCellByPosition(offset, unitcell, true);

        if (cell != null)
        {
            variants.Add(cell);
        }
    }

    foreach(Vector2 offset in attackvariants)
    {
        cell = FindCellByPosition(offset, unitcell, false);

        if (cell != null)
        {
            variants.Add(cell);
        }
    }

    return variants;
}

Ссылок: 2
public Cell FindCellByPosition(Vector2 offset, Cell UnitCell, bool MoveVariant)
{
    if (MoveVariant)
    {
        return GridRepository.Cells.Find(cell => cell.Position == UnitCell.Position + offset && cell.Unit == null);
    }
    else
    {
        return GridRepository.Cells.Find(cell => cell.Position == UnitCell.Position + offset && cell.Unit != null && cell.Unit.UnitSide != UnitCell.Unit.UnitSide);
    }
}
```

Рисунок 3.7 – Методи прогнозування варіантів ходу

Юніти можуть поділятися на різні види, кожен з них має свої характеристики, здатність переміщуватись по полю та атакувати клітинку ворога в залежності від дальності його атаки.

```
[SerializeField] private GridIteract _gridIteract;

[HideInInspector] public List<Cell> Variants = new List<Cell>();

public List<Vector2> MoveVariants;
public List<Vector2> AttackVariants;

public Side UnitSide;

public int Health = 10;
public int Damage = 2;

public void GoToCell(Cell cell)
{
    if (cell.Unit == null)
    {
        Move(cell);
    }
    else
    {
        if (DamageUnit(cell.Unit))
        {
            Move(cell);
        }
    }

    _gridIteract.DeselectCell(cell);
    _gridIteract.ClearVariants(Variants);
}
private void Move(Cell cell)
{
    _gridIteract.GridRepository.SelectedCell.Unit = null;

    transform.position = new Vector3(cell.transform.position.x, transform.position.y, cell.transform.position.z);

    cell.Unit = this;
    _currentCell = cell;
}
```

```
private bool DamageUnit(UnitBase enemy)
{
    enemy.Health -= Damage;

    if (enemy.Health <= 0)
    {
        enemy.Die();

        return true;
    }

    return false;
}

ссылка: 1
private void Die()
{
    _currentCell.Unit = null;

    Destroy(gameObject);
}
```

Рисунок 3.8 – Методи переміщення та атаки юніту

Візуальне середовище гри представлено, як локація з декораціями в стилі Low Poly, що містить бойове поле та розміщених на ньому юнітів.

Рисунок 3.9 – Демонстрація геймплею

Рисунок 3.10 – Демонстрація геймплею

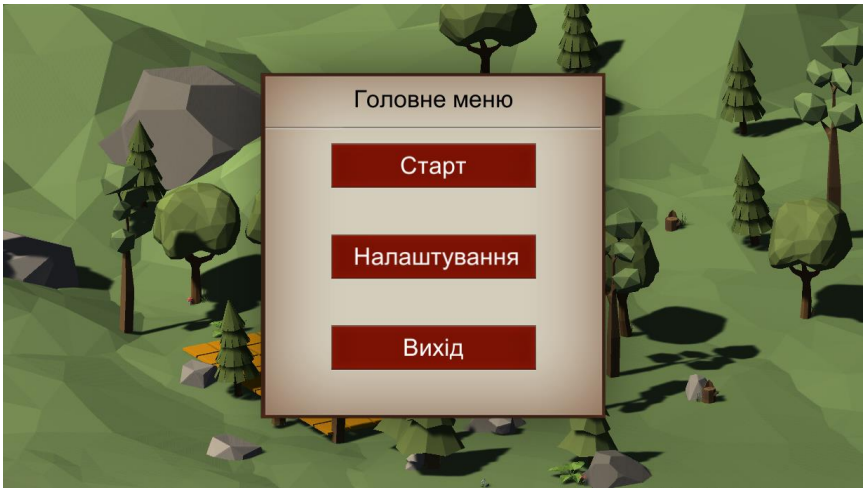


Рисунок 3.11 – Головне меню гри

4 ОГЛЯД ВИКОРИСТАНИХ ІНСТРУМЕНТІВ ДЛЯ СТВОРЕННЯ ІГОР

4.1 Visual Studio

Рисунок 4.1 – логотип Visual Studio

Програмне забезпечення компанії Microsoft є популярним вибором розробників в написанні коду для скриптів майбутніх проектів. Microsoft Visual Studio має обширний інструментарій та середовище розробки програмного забезпечення в яке входить редактор коду та інструменти для відлагодження програм та складання в автоматизованому режимі.

Visual Studio може мати від одного до кількох наступних компонентів: Visual Basic .NET, Visual C++, Visual C#, Visual F#, Visual Studio Debugger.

В Visual Studio входить редактор, налагоджувач, автозавершувач коду, компілятори, графічні конструктори та інші функції.

Популярні інструменти для підвищення продуктивності Visual Studio:

4.1.1 Меню швидких дій

Під час написанню коду коли допускаються помилки програмістом код в Visual Studio підкреслює хвилястою лінією місце в коді які можуть викликати помилку або вже являються нею, при наведенні курсором миші на підкреслене хвилястою лінією місце то з’явиться вікно з підказкою на джерело можливої проблеми ще поряд з’явиться лампочка те саме меню швидких дій в якому при розгортанні будуть запропоновані варіанти для виправлення помилки.

4.1.2 Очистка коду

Можна відформатовати код всього за один клік по створеним профілям форматування відступ, інтервали, шрифт, переноси та по бажанню виправить допущені помилки в коді якщо це можливо дана можливість доступна лише для коду на C#. Такі параметри налаштувань зберігаються у файлі під назвою «EditorConfig» можна зробити копію такого профілю щоб використати на іншому робочому місці.

4.1.3 Рефакторинг

Рефакторинг дає можливість перейменувати змінні якщо у вашому проекті з’являється необхідність змінити назву змінної а таких змінних п’ятдесят чи більше було б не продуктивно витрачати купу на таку діяльність тому була створена функція інтелектуального перейменування змінних яке в пару натискань перейменує хоч тисячу змінних швидко і зручно також можна перенести один чи кілька рядків коду в інший метод та зміна порядку параметрів методу.

4.1.4 IntelliSense

IntelliSense це такий набір можливостей який спрощує та прискорює роботу програміста, IntelliSense може вставляти фрагменти коду по вбудованих в неї списками завершення при введені ключового слова чи вже оголошеної раніше змінної. Також IntelliSense може при наведенні на ключові слова наприклад виводить список членів типу.

4.1.5 Пошук в Visual Studio

В Visual Studio дуже багато пунктів меню, дій, параметрів щоб було швидко та зручно шукати необхідні пункти меню, фрагменти коду чи функції в інтегроване середовище розробки для розробників був розроблений компонент пошуку який можна викликати комбінацією клавіш «CTRL+Q».

4.1.6 Live Share

Live Share надає можливість здійснювати налагодження и редагування проекту сумісно з кимось в реальному часі, можна моментально пред’явити доступ до свого проекту при цьому буде підтримка високого рівня безпеки. Також можна пред’явити доступ і до деяких інших можливостей локального комп’ютера.

4.1.7 Ієрархія викликів

У Visual Studio є вікно «ієрархія викликів» завдяки ньому можна подивитися який метод за яким викликається корисний інструмент в відлагодженні коду проекту так як таким чином можна знайти помилку чи уточнити чи не з’явиться в майбутньому якщо буде прийняте рішення щось змінити.

4.1.8 CodeLens

CodeLens знаходить зміни коду, зв’язані з кодом помилки, робочі елементи, посилання на код, модульні тести.

4.1.9 Перейти до визначення та показати визначення

Функція «перейти до визначення» дає можливість перейти до місця де знаходиться обрана функція або тип. У вікні «показати визначення» можна побачити метод або визначення типу навіть не відкриваючи окремий файл.

4.2 Unity

Рисунок 4.2 – логотип Unity

Unity – це середовище розробки ігор та додатків під різні платформи починаючи від персональних комп’ютерів до смарт годинників. Unity постачається в комплекті з редактором коду та редактором сцени на котрій можна налаштовувати поведінку об’єктів.

4.2.1 Редактор сцени

Наявна можливість редагування сцени прямо під час тестування увімкненої гри/програми, тобто якщо результат роботи якогось об’єкту не влаштовує то його можна в реальному часі змінити та відразу подивитися результат.

4.2.2 Тривимірне моделювання/UI інтерфейс

Unity надає можливість розробникам створювати тривимірні моделі по своєму вподобанню після створення і встановлення необхідних параметрів об’єкту можна буде подивитися як та буде поводити себе на сцені в різних умовах. Також в Unity можна створювати якісний звичний UI інтерфейси для зручності навігації користувачів в додатках. Також ніхто не забороняє використати вже готові моделі та UI інтерфейси розроблені в інших застосунках.

4.2.3 Сучасна графіка

Двигун Unity використовує сучасну графіку також має вбудований двигун рендерингу в реальному часі тому коли вносяться зміни в графіці вони одразу будуть застосовані і в вікні редактору будуть видимі зміни.

Також двигун Unity оснащений технологіями для роботи з глобальним освітленням, трасуванням променів та фізикою відображень ці технології допомагають створювати реалістичну картинку на дисплеях різних пристроїв. Та для того щоб ці технології працювали швидко та справно необхідна підтримка API популярних виробників відеокарт та технологій: Directx 12, Vulkan, iOS Metal, NVIDIA VRWorks та AMD LiquidVR.

4.2.4 Готова фізика та взаємодія предметів

Всі знають що камінь в повітрі сам по собі висіти не може і що від того на яку поверхню він (чи якийсь інший предмет з іншого матеріалу) впаде чи зіштовхнеться має відбутися дія зв’язана так чи інакше з фізикою щоб це стало можливо необхідно перенести дуже багато правил реального світу в код яких буде відтворювати машина на щастя розробникам майже ніколи не потрібно цього робити в Unity є написана фізика для різних об’єктів також є шаблони для створення індивідуальних правил поведінки об’єктів.

4.2.5 Скрипти на C#

Раніше Unity позволяла писати скрипти на двох мовах на вибір це JavaScript та C# наразі лише останній з причини що дуже малий відсоток проектів писалися на JavaScript і подальша його підтримка була не вигідною для компанії. За допомогою скриптів можна робити дуже велику кількість речей від створення/змінення об’єктів до зміни правил ігрового середовища. Можна намалювати інтерфейс створити діючих персонажів та середовище в якому має відбуватися якесь дійство та без скриптів нічого не буде відбуватися проте за допомогою скриптів розробник може зробити майже все що собі навигадує.

4.2.6 Безкоштовно

Unity поширюється на безкоштовній основі її можна просто завантажити з офіційного сайту розробника та використовувати в своїх цілях будь це розробка якогось застосунку чи гри та є не велике обмеження розробник не може отримувати інвестиції розміром більше ста тисяч доларів відсутня командна робота над проектом та закритий початковий код. Ці обмеження не стануть на дорозі до вивчення роботи в Unity чи навіть до становлення інді розробником який зможе забезпечити собі життя.

4.2.7 Мультиплатформність

Мультиплатформність це коли програма працює, поширюється більше ніш на одній платформі такий підхід суттєво зменшує кошти при переносі на нову платформу та затрачений час на адаптацію.

Рисунок 4.3 – Платформи які підтримує Unity

Двигун Unity підтримує більше 26 платформ і однакові логотипи являються різними платформами від однієї компанії.

4.2.8 Unity Asset Store

Unity Asset Store являється магазином у якому можна придбати ассети для майбутньої гри це можуть бути 3D моделі, звуки\музика, UI елементи, шейдери,

інструменти та спрайти. В асортименті магазину є як платні ассети так і безкоштовні тому навіть якщо розробник не має таланту чи знайомого який би зробив гарні моделі чи написав музику та інше це не повинно стати перепоною для створення хорошої гри.

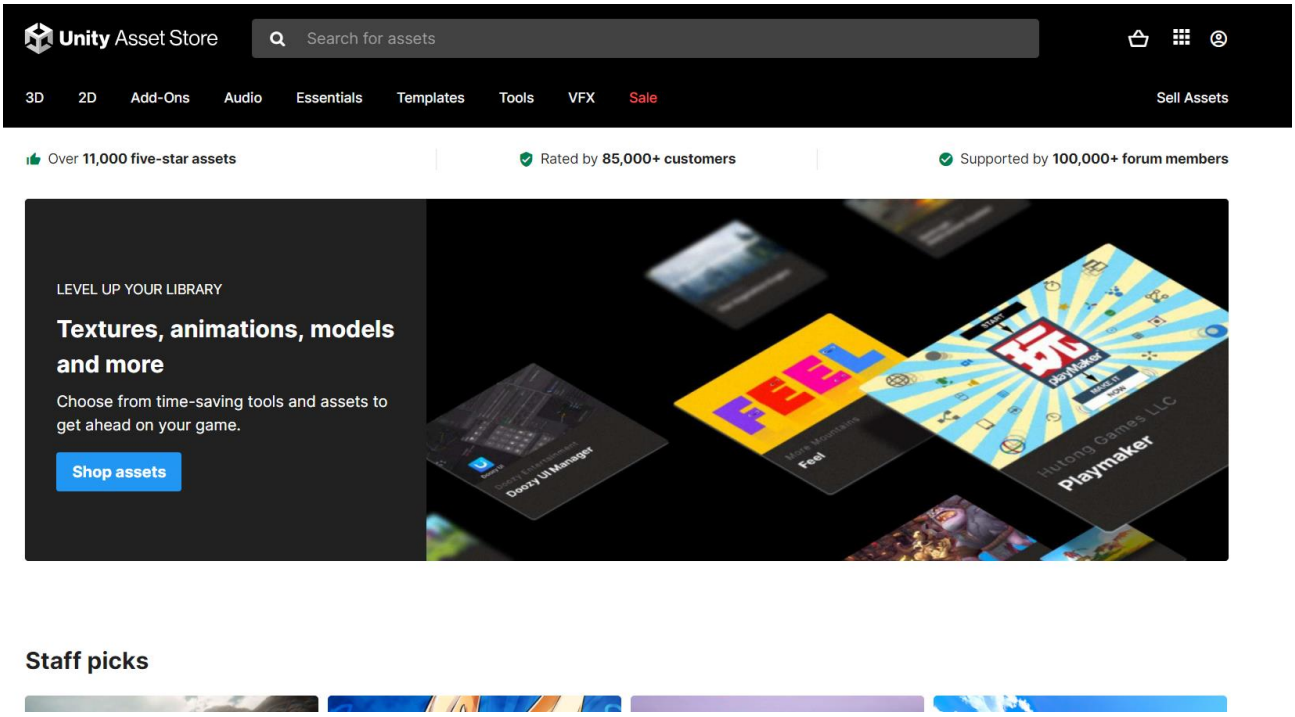


Рисунок 4.4 – Головна стрінка Unity Asset Store

4.2.9 ScriptableObject

ScriptableObject це клас який дає можливість створювати об'єкти типу Scriptable Object щоб зберігати велику кількість даних. ScriptableObject в Unity є дуже корисним і зручним інструментом для збереження великої кількості даних в одному або декількох відсортованих місцях що не тільки допомагає легше знайти необхідні дані у випадку необхідності а і може зменшити об'єм використаної пам'яті для копій префабу це стає можливо через об'єднання спільних даних наприклад ліс з дерев замість того аби завантажувати модель кожного дерева по одному для кожної копії завантажуються одразу всі так як вони однакові тільки позиція буде різна та таким чином буде використано менше ресурсів пристрою та швидше.

4.2.10 Ассети

Це компоненти без яких створення застосунків було б не можливе так як вони представляють собою музичний супровід, скрипти, графіку, 3D моделі та інше. Компоненти кріпляться до різних об’єктів і являються важливою частиною кожного проекту.

Наприклад, дерево на сцені до нього можуть бути прикріплені такі компоненти:

- 3D модель дерева;
- Анімації покачування від вітру, дрибижання при попаданні по ньому, падіння при зрублені;
- Звуки листя, тріщання кори;
- Скрипти відповідальні за перехід стадій анімацій або як довго дерево вистоїть від ударів топору.

Все це необхідно аби дерево не виглядало як білий витягнутий у верх циліндр, а щоб дерево на сцені видавалося схожим на те реальне дерево яке люди кожного дня бачать.

Які бувають ассети:

- спрайти - двовимірні зображення ігрових об'єктів (дерева, фон, персонажі);
- аудіо - музика та звукові ефекти;
- візуальні ефекти - вибухи, фільтри, вібрація та інше;
- моделі - тривимірні об'єкти (замінюють спрайти у тривимірних іграх);
- префаби - заготівлі об'єктів (готові об'єкти із прикріпленими компонентами);
- текстури – зовнішній вигляд для моделей;
- сцени – локації для гри;
- елементи інтерфейсу - шрифти, кнопки, зображення;

- анімації - спеціальні файли, у яких вказується порядок зміни кадрів для двовимірних ігор у тривимірних моделей це поряд переміщення координат моделі по заданому порядку;
- скрипти - ігровий інтелект, код керування персонажем та інше.

ВИСНОВКИ

В результаті виконаної дипломної роботи, розроблена гра в жанрі «покрокова стратегія» для підвищення зацікавленості гравців в ігровому процесі покрокової стратегії за рахунок реалізованого локального багатокористувацького режиму(підтримкою керування кількома ігровими акаунтами) де двоє гравців мають змогу позмагатися в стратегічному веденні битви з метою перемоги над опонентом.

Розроблена система генерації сітки дозволяє розмістити юнітів гравці по різні сторони на ігровому полі гри та щоб гравці могли переміщувати своїх юнітів по полю бою для здійснення стратегічних маневрів з метою отримання певної вигоди в подальшому.

Для розширення стратегічних маневрів та для більшої зацікавленості гравців було розроблено два різних юніти перший це воїн який може боротися тільки в ближньому бої та має більше очків здоров’я та арбалетник який має менший запас здоров’я в порівнянні з воїном та арбалетник має перевагу у відстані атаки таким чином наприклад: воїнами можна обороняти арбалетник або ж використати арбалетників як приманку що значно збільшує кількість стратегічних підходів до ведення бою.

Розроблено UI інтерфейс головного меню з якого гравці можуть розпочати гру, перейти в налаштування щоб відрегулювати гучність звуку та вийти з гри коли вони того забажають. Також після початку гри у гравців на екрані відображатиметься кнопка завершення ходу що передати хід наступному гравцю коли він закінчить свій. Після того як остання фігура одного з гравців буде переможена на екран буде виведено вікно з результатом бою в якому буде відображено який гравець переміг в битві та питання чи бажають гравці зіграти ще, якщо так то гра почнеться з самого початку, якщо ні відбудеться повернення до головного меню.

ПЕРЕЛІК ПОСИЛАНЬ

1. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://osvita.ua/vnz/reports/psychology/28614/ Дата звернення: 1.1.2022
2. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://habr.com/ru/post/589295/ Дата звернення: 1.1.2022
3. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://thecode.media/unity/ Дата звернення: 1.1.2022
4. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.theverge.com/2015/3/3/8142099/unity-5-engine-release Дата звернення: 1.1.2022
5. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://blog.unity.com/community/unityscripts-long-ride-off-into-the-sunset Дата звернення: 1.1.2022
6. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://docs.unity3d.com/Manual/class-ScriptableObject.html Дата звернення: 11.05.2022
7. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://habr.com/ru/post/421523/ Дата звернення: 11.05.2022
8. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://ain.ua/ru/2020/05/22/chto-takoe-dvizhok-unreal-engine/ Дата звернення: 14.05.2022
9. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://docs.microsoft.com/ru-ru/visualstudio/get-started/visual-studio-ide?view=vs-2022 Дата звернення: 15.05.2022
10. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://dut.edu.ua/ua/news-1-626-8002-zastosuvannya-uml-chastina-3-diagrama-klasiv----class-diagram_kafedra-kompyuternih-nauk-ta-informaciynih-tehnologiy Дата звернення: 29.05.2022

11. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://leopotam.com/5/ Дата звернення: 29.05.2022
12. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://docs.unity3d.com/Manual/Profiler.html Дата звернення: 30.05.2022
13. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://docs.unity3d.com/Manual/LowLevelNativePluginProfiler.html Дата звернення: 30.05.2022
14. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://help.briovr.com/hc/en-us/articles/360059775633-Light-Baking-Options Дата звернення: 30.05.2022
15. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.rastergrid.com/blog/2010/10/gpu-based-dynamic-geometry-lod/ Дата звернення: 30.05.2022
16. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.codeproject.com/Articles/330742/Texture-Atlas-Maker Дата звернення: 30.05.2022
17. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://unity.com/features/shader-graph Дата звернення: 30.05.2022
18. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://sourcemaking.com/design_patterns/object_pool Дата звернення: 30.05.2022

ДОДАТКИ

1. Лістинг

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

public class GridIteract : MonoBehaviour
{
    public GridRepository GridRepository;

   public void SelectCell (Cell cell)
    {
        GridRepository.SelectedCell = cell;
        cell.State = State.Selected;
        cell.ChangeColor(cell.SelectedColor);

        if (cell.Unit != null)
        {
            cell.Unit.Variants = CalculateVariants(cell, cell.Unit.MoveVariants,
cell.Unit.AttackVariants); //пошук ходів

            ShowVariants(cell.Unit.Variants);
        }
    }

    public void DeselectCell(Cell cell)
    {
        GridRepository.SelectedCell.State = State.Standart;
        GridRepository.SelectedCell.ChangeColor(GridRepository.SelectedCell.StandartColor);
/*зміна назв кольорів*/
        GridRepository.SelectedCell = null;

        if (cell.Unit != null)
        {
            ClearVariants(cell.Unit.Variants); //почистити варіанти ходів
            cell.Unit.Variants.Clear();
        }
    }

    public List<Cell> CalculateVariants(Cell unitcell, List<Vector2> movevariants,
List<Vector2> attackvariants)
    {
        var variants = new List<Cell>();
        Cell cell;

        foreach (Vector2 offset in movevariants)
        {
            cell = FindCellByPosition(offset, unitcell, true);

            if (cell != null)
            {
                variants.Add(cell);
            }
        }

        foreach(Vector2 offset in attackvariants)
        {
            cell = FindCellByPosition(offset, unitcell, false);
```

```
            if (cell != null)
            {
                variants.Add(cell);
            }
        }

        return variants;
    }

    public Cell FindCellByPosition(Vector2 offset, Cell UnitCell, bool MoveVariant)
    {
        if (MoveVariant)
        {
            return GridRepository.Cells.Find(cell => cell.Position == UnitCell.Position +
offset && cell.Unit == null);
        }
        else
        {
            return GridRepository.Cells.Find(cell => cell.Position == UnitCell.Position +
offset && cell.Unit != null && cell.Unit.UnitSide != UnitCell.Unit.UnitSide);
        }
    }

    public void ShowVariants(List<Cell> variants) /*Відображення варіатів ходу, аргумент
список з збереженими варіантами ходу*/
    {
        foreach (Cell variant in variants)
        {
            variant.State = State.Variant;

            if (variant.Unit == null)
            {
                variant.ChangeColor(variant.MoveColor);
            }
            else
            {
                variant.ChangeColor(variant.AttackColor);
            }
        }
    }

    public void ClearVariants(List<Cell> variants) /*аргумент варіантс з поточними варіантами
ходу*/
    {
        foreach (var variant in variants)
        {
            variant.State = State.Standart;
            variant.ChangeColor(variant.StandartColor);
        }
    }
}


using System.Collections.Generic;
using UnityEngine;

public enum Side
{
    Player,
    Enemy
}

public class UnitBase : MonoBehaviour
{
    public List<Vector2> MoveVariants;
```

```
    public List<Vector2> AttackVariants;

    public Side UnitSide;

    public int Health = 10;
    public int Damage = 2;

    [SerializeField] private Cell _currentCell;

    [SerializeField] private GridIteract _gridIteract;

    [HideInInspector] public List<Cell> Variants = new List<Cell>();


    public void GoToCell(Cell cell)
    {
        if (cell.Unit == null)
        {
            Move(cell);
        }
        else
        {
            if (DamageUnit(cell.Unit))
            {
                Move(cell);
            }
        }

        _gridIteract.DeselectCell(cell);
        _gridIteract.ClearVariants(Variants);
    }

    private void Move(Cell cell)
    {
        _gridIteract.GridRepository.SelectedCell.Unit = null;

        transform.position = new Vector3(cell.transform.position.x, transform.position.y,
cell.transform.position.z);

        cell.Unit = this;
        _currentCell = cell;
    }

    private bool DamageUnit(UnitBase enemy)
    {
        enemy.Health -= Damage;

        if (enemy.Health <= 0)
        {
            enemy.Die();

            return true;
        }

        return false;
    }

    private void Die()
    {
        _currentCell.Unit = null;

        Destroy(gameObject);
    }

    [ContextMenu("Initialize")]
```

```
    public void InitializeCell()
    {
        Ray ray = new Ray(transform.position, Vector3.down);
        RaycastHit hit;

        if (Physics.Raycast(ray, out hit, 1000f))
        {
            if (hit.collider.GetComponent<Cell>())
            {
                transform.position = new Vector3(hit.transform.position.x,
transform.position.y, hit.transform.position.z);

                hit.collider.GetComponent<Cell>().Unit = this;

                var cell = hit.collider.GetComponent<Cell>();

                _gridIteract = cell.GridIteract;
                _currentCell = cell;
                _gridIteract = cell.GridIteract;
            }
        }
    }
}


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

public class GridRepository : MonoBehaviour
{
    public List<Cell> Cells = new List<Cell>(); /*для збереження клітинок*/
    public Cell SelectedCell; /*для виділеної клітинки*/
}


using UnityEngine;

public enum State /*стадії клітки*/
{
    Standart,
    Selected,
    Variant
}
public class Cell : MonoBehaviour
{
    public Color StandartColor; /*стандартний колір клітки*/
    public Color HoverColor; //колір клітинки при наведенні
    public Color HoverMoveColor; /*колір клітинки коли по ній можна пройти*/
    public Color HoverAttackColor; /*колір клітинки коли на ній ворог в діапазоні атаки*/

    public Color SelectedColor;
    public Color MoveColor;
    public Color AttackColor;

    public MeshRenderer MeshRenderer;

    public UnitBase Unit; /*юніт на клітинці*/
    [HideInInspector] public Vector2 Position; /*для зміни позиції клітинки*/
    [HideInInspector] public State State; /*для збереження стану клітинки*/

    public GridIteract GridIteract;

    public void ChangeColor(Color color) //метод для зміни кольору
    {
```

```
        MeshRenderer.material.color = color;
    }

    private void OnMouseEnter()
    {
        if (State == State.Standart)
        {
            ChangeColor(HoverColor); /*зміна кольору клітинки принаведенні миші*/
        }
        else if (State == State.Variant)
        {
            if (Unit == null)
            {
                ChangeColor(HoverMoveColor);
            }
            else
            {
                ChangeColor(HoverAttackColor);
            }
        }
    }

    private void OnMouseDown() /*при натисканні на клітинку*/
    {
        if (State == State.Selected)
        {
            GridIteract.DeselectCell(this); //прибрати виділення
        }
        else if (State == State.Variant)
        {
            GridIteract.GridRepository.SelectedCell.Unit.GoToCell(this);
        }
        else if (State == State.Standart)
        {
            if (GridIteract.GridRepository.SelectedCell != null)
            {
                GridIteract.DeselectCell(GridIteract.GridRepository.SelectedCell);
            } //виділяти клітинку

            GridIteract.SelectCell(this);
        }
    }

    private void OnMouseExit()
    {
        if (State == State.Standart)
        {
            ChangeColor(StandartColor); /*повернення стандартного кольору коли клітинка не
виділена*/
        }
        else if (State == State.Variant)
        {
            if (Unit == null)
            {
                ChangeColor(MoveColor);
            }
            else
            {
                ChangeColor(AttackColor);
            }
        }
    }
}
```

```
using UnityEngine;

public class GridGenerator : MonoBehaviour
{
    [SerializeField] private Vector2Int _gridSize; /*довжина сітки по x,y*/
    [SerializeField] private Cell _cellPrefab; /*префаб клітинки*/
    [SerializeField] private float _offset; /*відступ між клітинками*/
    [SerializeField] private Transform _parent; /*обєкт на якому зберігаються обєкти*/
    [SerializeField] private GridRepository _gridRepository;
    [SerializeField] private GridIteract _gridIteract;

    [ContextMenu("Generate grid")]
    private void GenerateGrid() /*основна логіка генерації сітки*/
    {
        _gridRepository.Cells.Clear();
        _cellPrefab.GridIteract = _gridIteract;

        var cellSize = _cellPrefab.GetComponent<MeshRenderer>().bounds.size; /*отримуємо
розмір сітки*/

        for (int x = 0; x < _gridSize.x; x++) /*генерація самої сітки*/
        {
            for (int y = 0; y < _gridSize.y; y++)
            {
                var position = new Vector3(x * (cellSize.x + _offset), 0, y * (cellSize.z +
_offset)); /*позиція клітинки з урахуванням відступів*/

                var cell = Instantiate(_cellPrefab, position, Quaternion.identity, _parent);
//обєкт клітинки

                cell.name = $"X: {x} Y: {y}";
                cell.Position = new Vector2(x, y);

                _gridRepository.Cells.Add(cell);
            }
        }
    }
}


public class Solider : UnitBase
{

}
```

Демонстраційні матеріали

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра Інженерії програмного забезпечення

РОЗРОБКА ГРИ "THE LOSS OF BAGRASH"В ЖАНРІ ПОКРОКОВА СТРАТЕГІЯ НА ДВИГУНUNITY

Виконавець: студент 4 курсу,
групи ПД-43
Музика Антон Павлович
Керівник роботи: доктор філософії (PhD)
Дібрівний Олесь Андрійович

Київ 2022

Аналіз аналогів

Категорії порівняння	The Banner Saga	Ach Of Gods	The Art of Tahira	The Loss Of Bagrash
Сюжет	+	+	+	-
Можливіть гри разом на одному пристрої	-	-	-	+
Підтримка української мови	-	+	-	+
Можливість зіграти самому	+	+	+	+

Мета, об’єкт та предмет роботи

Мета роботи – розвиток логічного мислення гравців за рахунок розробки гри в жанрі «покрокова стратегія» з можливостю грати у двох на одному ПК

Об'єкт дослідження– ігри в жанрі «покрокова стратегія».

Предмет дослідження– вплив покрокових стратегій на критичне мислення та планування дій.

3

Технічне завдання

1. Розробити систему генерації сітки для бою та переміщення.
2. Впровадити взаємодію гравця з ігровою сіткою.
3. Реалізувати спосіб переміщення ігрових фігур по сітці.
4. Розробити юнітів з різним варіантами атаки.
5. Розробити меню.
6. Змоделювати візуальне середовище.

4

Засоби розробки застосунку

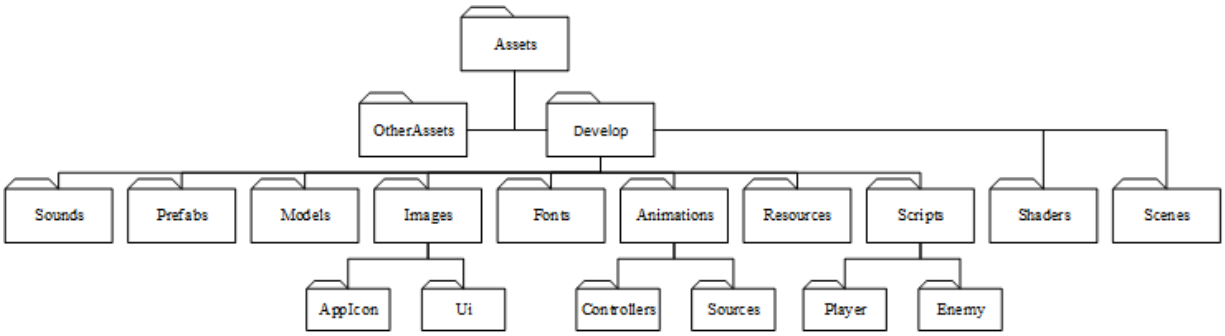
Visual Studio

5

Діаграма варіантів використання гравця

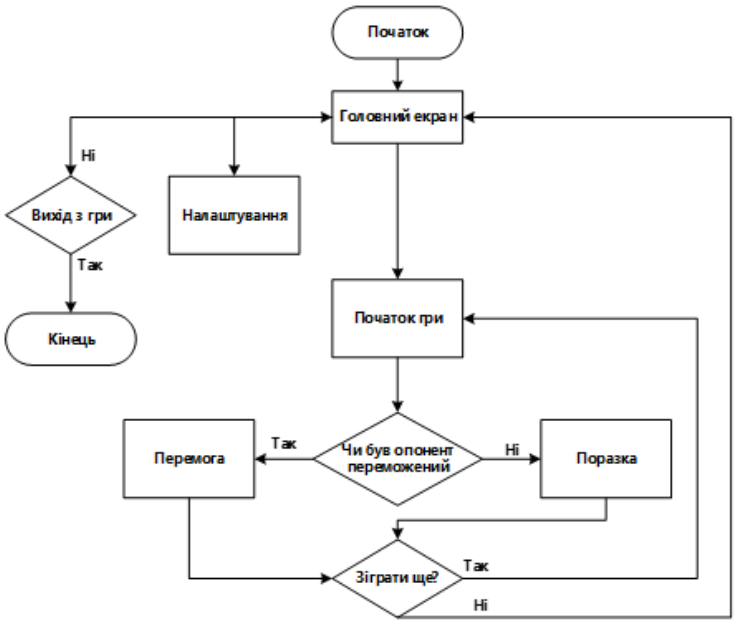
6

Структура застосунку



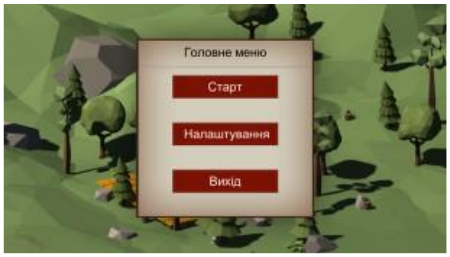
7

Алгоритм роботи застосунку



8

Екранні форми



Головне меню

Демонстрація геймплею№1

Демонстрація геймплею №2

9

Апробація результатів дослідження

Музика А.П., ScriptableObject – що це таке / Науково-технічна конференція «Застосування програмного забезпечення в інфокомунікаційних технологіях». Збірник тез 20 квітня 2022, ДУТ, м. Київ-К.: ДУТ, 2022. С.101-102.

10

Висновки

Створено гру на двигуніUnity в жанрі «покрокова стратегія» в якій двоє гравців мають можливість зіграти один проти одного на одному персональному комп'ютері, для цього було виконано наступні завдання:

1. Розроблено систему генерації сітки для бою та переміщення.
2. Впроваджено взаємодію гравця з ігровою сіткою.
3. Реалізовано спосіб переміщення ігрових фігур по сітці.
4. Розроблено юніти з різними варіантами атаки.
5. Розроблено меню.
6. Змодельовано візуальне середовище.

Дякую за увагу!