ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи
На ступінь вищої освіти бакалавр
на тему:
«Розробка гри жанру стратегія у реальному часі на мові програмування C# за допомогою ігрового двигуна Unity»

Виконав: студент 4 курсу, групи ПД-44
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
Сокирко Д.О.
(прізвище та ініціали)
Керівник Дібрівний О.А.
(прізвище та ініціали)
Рецензент _____
(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти – «Бакалавр»
Напрям підготовки – 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
Негоденко О.В
“ ____ ” _____ 2022 року

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

Сокирку Дмитру Олександровичу
(прізвище, ім’я, по батькові)

1. Тема роботи: Розробка гри жанру стратегія у реальному часі на мові програмування C# за допомогою ігрового двигуна Unity
Керівник роботи Дібрівний О.А., доктор філософії, доцент _____,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від “18” лютого 2022 року №__.
2. Строк подання студентом роботи 03.06.2022
3. Вхідні дані до роботи:
   3.1 Технічна документація
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
   4.1 Розробка комп’ютерної гри у жанрі стратегія у реальному часі, використовуючи мову програмування C# за допомогою ігрового рушія Unity та середи розробки Visual Studio для роботи з мовою
   4.2 Опис та класифікація предметної області, визначення функціоналу та алгоритму проекту
   4.3 Аналіз існуючих ігор у цьому жанрі
   4.4 Дослідження рушіїв
   4.5 Визначення функціоналу та алгоритму проекту
5. Перелік графічного матеріалу:
   5.1 Струтурна схема алгоритму

5.2 Діаграма класів

5.3 Висновки

6. Дата видачі завдання 11.04.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	11.04 - 14.04	
2	Вивчення та аналіз існуючих прототипів	15.04 - 17.04	
3	Розробка вимог та оцінка якості	18.04 - 21.04	
4	Розробка гри	22.04 - 25.04	
5	Вступ, висновки, реферат	07.05 – 10.05	
6	Розробка презентації	11.05 – 15.05	
7	Попередній захист роботи	16.05 – 02.06	
8	Здача роботи	03.06	

Студент _____ Сокирко Д.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Дібрівний О.А
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 71 с., 27 рис., 1 табл., 18 джерел.

ГРА, UNITY, РОЗРОБКА, СТРАТЕГІЯ, СТРАТЕГІЯ У РЕАЛЬНОМУ ЧАСІ, RTS, РУШІЙ, ДВИГУН, C#, VISUAL STUDIO.

Об’єкт дослідження –ігровий процес ігор жанру стратегія у реальному часі.

Предмет дослідження – програмні засоби для розробки гри жанру стратегія у реальному часі.

Мета роботи – привернути увагу нових гравців до ігор жанру стратегія у реальному часі.

У ході виконання роботи було розглянуто та проведено аналіз вже існуючих ігор даного жанру. Створено порівняльну характеристику, яка порівнює переваги та недоліки даних продуктів.

Для досягнення поставленої мети даної роботи було:

- Проведено аналіз ігрового рушія Unity, його компонентів та можливостей, та встановлено, що вибір саме цього ігрового рушія є вдалим рішенням;
- Розроблено алгоритми для реалізації комп’ютерної гри жанру стратегія у реальному часі за допомогою ігрового двигуна Unity;
- Взявши до уваги усі проведені дослідження та перевірки було розроблено комп’ютерну гру у жанрі стратегія у реальному часі.

Галузь використання – завдяки вільному доступу до гри будь-хто зможе випробувати дану гру та ознайомитися з доволі цікавим жанром стратегій.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

UI – user interface

GUI – graphical user interface

UI – user interface

AI – artificial intelligence

RTS – real time strategy

ОС – операційна система

ПЗ – програмне забезпечення

ПК – персональний комп’ютер

Юніт – мирна або військова одиниця задіяна у стратегічних відеоіграх

ВСТУП

Актуальність дослідження. На сьогоднішній день важко знайти людину, яка б не чула що таке відеоігри, вони стрімко увірвалися у людське життя та посіли у ньому важливу нішу. Безліч людей використовують відеоігри для багатьох потреб, дехто просто щоб відпочити та приємно провести час, дехто щоб перевірити свої навички та здібності у хардкорних режимах, а інші навіть влаштовують міжнародні турніри або локальні змагання, щоб визначити хто з них найкращий у тій чи іншій сфері. Різноманіття жанрів та стилей дозволяє кожному знайти те що прийдеться йому до душі.

Далеко також дійшла і сфера розробки відеоігор, все більше і більше розробників бажають займатися саме відеоіграми. На сьогоднішній день існує багато великих ігрових компаній таких як Activision Blizzard, Ubisoft Entertainment, Electronic Arts та інших, та ще набагато більше невеличких студій чи груп людей, які присвятили себе цій індустрії.

Також великим поштовхом до розвитку індустрії відеоігор став стрімкий розвиток комп’ютерів та їх компонентів, що дозволило вивести ігри на новий рівень. Зараз багато ігор мають настільки деталізовану візуальну складову, що їх навіть іноді важко відрізнити від реальності.

Мета та завдання роботи. Метою роботи являється розробка гри жанру стратегія у реальному часі за допомогою мови програмування C# та ігрового двигуна Unity.

Задля досягнення цієї мети потрібно вирішити деякі завдання, а саме:

- Провести порівняльну характеристику вже існуючих ігор подібного жанру, визначити їх переваги та недоліки;

- Переглянути та вивчити теоретичні відомості щодо розробки ігор даного жанру та розробки ігор в цілому, обрати найкращі шляхи досягнення поставлених цілей;
- Описати базовий функціонал майбутнього продукту;
- Розробити основні модулі комп’ютерної гри;
- Провести тестування готового продукту.

Об’єкт та предмет дослідження. Об’єктом дослідження виступає розробка комп’ютерних ігор.

Предметом дослідження є розробка ігор жанру стратегія у реальному часі за допомогою ігрового рушія Unity.

Методи дослідження. Під час розробки даного продукту було використано такі методи як: метод аналогії, порівняння, експерименту та аналізу.

Практичне значення. Практичним значенням даного дослідження являється розробка гри стратегія у реальному часі за допомогою доступних технологій та з вирішенням основних проблем та недоліків даного жанру.

1 АНАЛІЗ ОБЛАСТІ ДОСЛІДЖЕНЬ

1.1.Аналіз ігрових платформ

Паралельно з розвитком відеоігор розвиваються і платформи на яких у них грають. Платформи бувають як такі, які можна використовувати в дорозі, так і статичні, які мають залишатися у певному місці. Наразі існує доволі велика конкуренція між комп'ютерними та консольними іграми, що ще більше прискорює їх розвиток. І хоча ПК та консолі тримають першість у популярності, існує також і багато інших платформ, які впевнено тримають планку та можливо у майбутньому навіть наздоженуть своїх конкурентів.(Рис. 1.1)

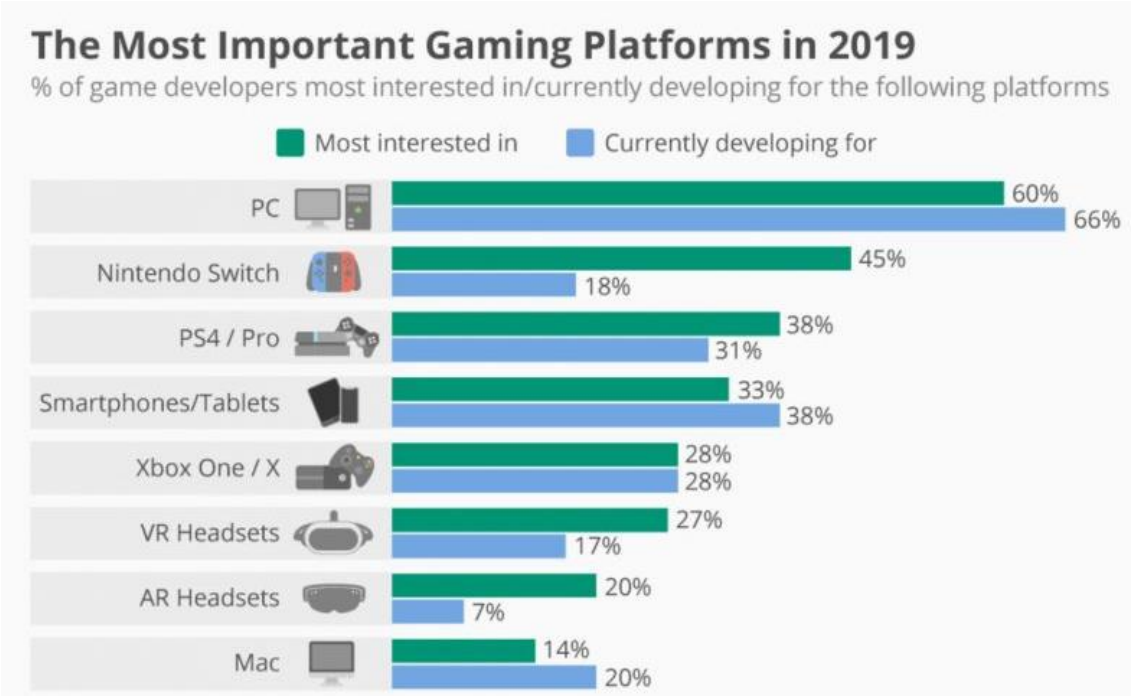


Рис. 1.1 Таблиця популярності ігрових платформ

1.1.1 ПК

Найбільш популярною платформою для ігор наразі являється ПК. Їх популярність зумовлена багатьма причинами, деякими із них являється більш «сильніше» залізо, що дозволяє отримати краще зображення та продуктивність. Також однією з важливих переваг комп’ютерів є можливість більш гнучкого налаштування як самої гри так і аксесуарів для керування грою. Також для ПК існують досить популярні ексклюзивні проекти, такі як World of Warcraft та League of Legend.

1.1.2 Консолі

Велику популярність у індустрії відеоігор також отримали ігрові консолі. Їхня популярність зумовлена легкістю у використанні, наявністю багатьох ексклюзивних ігор та доволі привабливий дизайн. Однією з найбільш привабливих консолей являється консоль від PlayStation, а саме PlayStation 2, яка до 2019 року була продана у кількості 159 мільйонів одиниць. Вона являється настільки популярною завдяки багатьом інноваціям компанії Sony, а також прихильністю задовольняти потреби геймерів.

1.1.3 Мобільні девайси

Останнім трендом сьогодні являється розробка мобільних ігор, адже у наш час смартфони стали невід’ємною частиною життя і майже кожна людина користується ними. Мобільні ігри отримують свою популярність завдяки тому, що у них можна зіграти у будь-якому місці та у будь-який час.

1.2 Жанри комп’ютерних ігор

На сьогоднішній день існує доволі багато жанрів відеоігор, та ще більше їх під жанрів, це зумовлено тим, що розробники намагаються створити щось нове та унікальне. Кожен може знайти той жанр, який йому до вподоби.

Основні жанри відеоігор:

- Пісочниця – це жанр відеоігор, у якому гравець має повну свободу дій, відкрите середовище та нелінійний ігровий процес;
- Стратегія – цей жанр відеоігор пішов від звичних усім настільних ігор, таких як шахи, шашки та інші. Він вимагає від гравців використовувати розроблену ними стратегію та тактику задля досягнення поставлених цілей. Цей жанр поділяється на такі піджанри:
    - Стратегія у реальному часі(RTS – real-time strategy)
    - Тактика у реальному часі(RTT – real-time tactics)
    - Багатокористувацька онлайн-бойова арена(MOBA – multiplayer online battle arena)
    - Захист вежі(Tower Defense)
    - Покрокова стратегія(TBS – turn based strategy)
    - Wargame
- Екшн – це ігри де гравець знаходиться у центрі подій, та повинен подолати поставлені перед ним перешкоди, цей жанр на сьогодні являється найпопулярнішим. Вони поділяються на такі піджанри:
    - Платформер(Platformer)
    - Шутер(Shooter)
    - Файтинг(Fighting)
    - Beat-em up
    - Стелс (Stealth)

    - Ритм ігри(Rhythm)
    - Виживання(Survival)
- Пригодницькі ігри – це жанр ігор, у якому гравець повинен взаємодіяти зі своїм оточенням задля вирішення різноманітних головоломок та розвитку історії. Жанр поділяється на такі піджанри:
    - Текстові пригоди(Text adventure)
    - Графічні пригоди(Graphic adventure)
    - Інтерактивні фільми(Interactive movie)
    - Візуальні новели(Visual novel)
    - 3D у режимі реального часу(Real-time 3D)
- Рольові ігри, або ж RPG – ведуть своє походження від Dungeon & Dragons, та інших рольових ігор на ручці і папері. Поділяється на:
    - Action RPG
    - MMORPG
    - Roguelikes
    - Tactical RPG
    - Sandbox RPG
- Спортивні ігри – ігри, які моделюють різні види спорту, від футболу до автомобільних перегонів. Складається з таких піджанрів:
    - Перегони(Racing)
    - Командний спорт(Team sport)
    - Змагання(Competitive)
    - Спортивні бої(Sport-based fighting)

Цей список можна продовжувати ще досить довго, але й так зрозуміло, що різноманітність відеоігор не може не вражати.

1.3 Стратегії у реальному часі

Стратегія у реальному часі (real-time strategy, RTS) – це жанр стратегічних відеоігор, у яких геймплей не поділений на окремі обмежені у часі ходи, як у покрокових стратегіях, і всі гравці діють одночасно. Стратегічну гру в режимі реального часу також можна назвати симуляцією в режимі реального часу або військовою грою в режимі реального часу.

Стратегії у реальному часі також поділяються за геймплеєм на глобальні, тактичні, економічні та інші. Тому окрім відсутності черговості ходів та можливості гравцям діяти одночасно пошук якихось інших особливостей може бути доволі суб’єктивним. За часту вважають, що RTS може бути лише та гра, яка обов’язково включає елементи будівництва бази, збору ресурсів та найму юнітів, але це стосується лише класичних RTS, з часом вони достатньо видозмінювалися і привносили багато нового, відмовляючись від звичних канонів. Наприклад у деяких іграх могли відмовитися від системи будівництва бази, дозволяючи викликати юнітів безпосередньо на ігрове поле, або ж взагалі обмежуючи їх кількість лише доступними у початку завдання.

Але не дивлячись на усі нововведення найбільш популярними все ж залишаються стратегії, у яких гравець має зосереджуватись на зборі ресурсів, створенні одиниць та перемоги над супротивником. У таких іграх гравці повинні збирати ресурси, та раціонально їх використовувати для розвитку бази та найму юнітів, задля подальшого нападу на ворогів.

Більшість класичних стратегій у реальному часі мають такі елементи:

- Збір ресурсів – це накопичення спеціальних ресурсів за допомогою юнітів, споруд або ж спеціальних ресурсних точок, задля подальшого їх використання.
- Будівництво бази – це створення спеціальних споруд, завдяки яким можливо винаймати юнітів, проводити дослідження та вдосконалення, або ж видобувати ресурси. Однією з особливостей будівництва є відносність часу створення споруд, яка може складати лічені секунди. У деяких стратегіях відсутня концепція побудови бази, але зазвичай це являється однією з основних механік.
- Створення армії – це найм спеціальних одиниць, які в подальшому будуть використовуватися задля атаки супротивника, або ж захисту власної бази.
- Атака противника – зазвичай являється однією з основних задач жанру, коли гравець використовує найнятих їм юнітів задля нападу та знищення юнітів або бази ворога.

1.4 Огляд ігор аналогів

Жанр стратегій у реальному часі почав зароджуватися ще у 1980-х роках, і за весь час їх існування з’явилося чимало визначних представників даного жанру. Далі будуть представлені найвідоміші із них.

1.4.1 Dune II

Ця гра являється одним із перших представників даного жанру відеоігор, вона була розроблена та видана у 1992 році для MS-DOS.

Ігровий процес являється звичним для жанру. Дія відбувається на прямокутній карті, яка вкрита туманом війни. Основною метою гри є знищення ворожої бази, попередньо зібравши юнітів для її атаки.(Рис. 1.2) У грі присутні 3 протиборствуючі фракції, а саме дім Атрідів, дім Харконенів та дім Ордосів, кожна сторона має приблизно однаковий набір споруд та юнітів, за виключенням деяких унікальних. Саме ця гра привнесла у жанр основні риси, а саме кілька проборчих сторін, «дерево технологій», структуру кампаній.

Рис. 1.2 Гра Dune II

1.4.2 Козаки

Козаки – це серія ігор жанру стратегія у реальному часі від Української компанії GSC Game World перша частина якої вийшла у 2001 році(Рис. 1.3).

Гра має основні риси притаманні жанру, але значно виділяється серед конкурентів великою кількістю типів ресурсів, ігрових сторін та унікальних механік. У грі присутні близько 20 націй з унікальними юнітами та спорудами. Однією з цікавих механік даної гри являється можливість захоплення деяких юнітів та споруд ворога, якщо вони не захищені бойовими одиницями, також у грі присутня вибіркова вразливість деяких одиниць до інших, так будинки та стіни можливо знищити лише тими юнітами які можуть їх атакувати.

Рис. 1.3 Cossacks Back to War

1.4.3 WarCraft 3 Reign of Chaos

WarCraft 3 – це гра розроблена студією Blizzard Entertainment у 2002 році та сюжетно являється прямим продовженням другої частини. У той час ця гра була однією з найочікуваніших, було зроблено близько 4.5 мільйонів пре замовлень та за місяць продано близько 1 мільйона копій. Гра отримала велику кількість гарних відгуків, а деякі видавництва також відмітили гру такими нагородами як «Краща гра року» та «Краща стратегія року».

Ігровий процес являється звичним для жанру(Рис. 1.4), але вніс деякі нововведення такі, як підтримка юнітів, що зі збільшенням кількості одиниць зменшує кількість видобуваємого золота, також було додано спеціальних юнітів «Героїв», які можуть піднімати свій рівень та використовувати спеціальні предмети.

Рис. 1.4 Warcraft 3

1.4.4 StarCraft 2

StarCraft 2 – це ще одна гра від студії Blizzard Entertainment, яка являється трилогією, перша частина якої вийшла 2010 року(Рис 1.5), друга 2013, а третя 2015, також у 2016 році вийшло ще одне офіційне доповнення.

У грі присутні три ігрові раси: терани, зерги та протоси, кожна з яких має свої повністю унікальні споруди та юнітів. Ця гра увібрала у себе усі найкращі риси стратегій у реальному часі та довела їх до досконалості. Також гарним доповненням стала бібліотека фізики Havoc для зображення динамічних тіней та взаємодій між об’єктами.

Рис 1.5 StarCraft 2 Wings of Liberty

1.5 Переваги та недоліки

Таблиця 1 Переваги та недоліки розглянутих ігор

Назва гри	Переваги	Недоліки
Dune II	Звук	Відсутність мережевої гри
	Цікавий геймплей	Застарілість
Cossacks	Різноманіття ігрових націй	Відсутність підтримки сучасного обладнання
	Реалістична фізика	
	Можливість керувати великою кількістю юнітів	Відсутність підтримки сучасного ПЗ
	Наявність багатьох типів ресурсів	
WarCraft 3	Сюжет	Застаріла графіка
	Велика кількість карт	
	Наявність редактору карт	
StarCraft 2	Сюжет	Висока ціна
	Візуальна складова	
	Ігротека, яка містить карти від інших гравців	

1.6 Висновок першого розділу

Провівши аналіз області дослідження, можна дійти до висновку, що хоча популярність даного жанру доволі невисока порівняно з минулими роками, все ж стратегії зайняли важливе місце у ігровій індустрії. Цей жанр не зник повністю, а лише чекає свого часу.

Отже підвівши підсумки можна дійти висновку, що хоч жанр переживає і не найкращі часи, досі знайдуться багато фанатів, які з радістю зацікавляться новою грою даного жанру.

2 АРХІТЕКТУРА ПРОЕКТУ

2.1 Ігровий рушій

З розвитком відеоігор розвивалися і середовища для їх розробки. Ігрові рушії – це центральна програмна частина відеоігор, яка відповідає за її технічну сторону та дозволяє у багато раз полегшити сам процес розробки відеогри. На сьогоднішній день існує доволі багато ігрових рушіїв, найбільш популярними з них являються Unreal Engine, Unity, GameMaker, RPG Maker.

Кожен ігровий рушій має свої як переваги так і недоліки. Далі буде розглянуто декілька основних ігрових рушіїв.

2.1.1 Unreal Engine

Unreal Engine – це ігровий рушій розроблений компанією Epic Games, який спочатку був створений для шутерів від першої особи, але згодом почав використовуватися і для створення ігор інших жанрів. Він може використовуватися не лише для створення відеоігор, а й для кіно і телебачення, або навіть як віртуальне середовище для дослідження та проектування(Рис. 2.1). Його плюсами є підтримка широкого спектру платформ, відкритий вихідний код та відсутність роялті за ігри доки дохід від них не перевищить 1 млн доларів.

За допомогою цього ігрового рушія було створено такі популярні серії ігор як: Bioshock, Mass Effect, Gears of War, та багато інших.

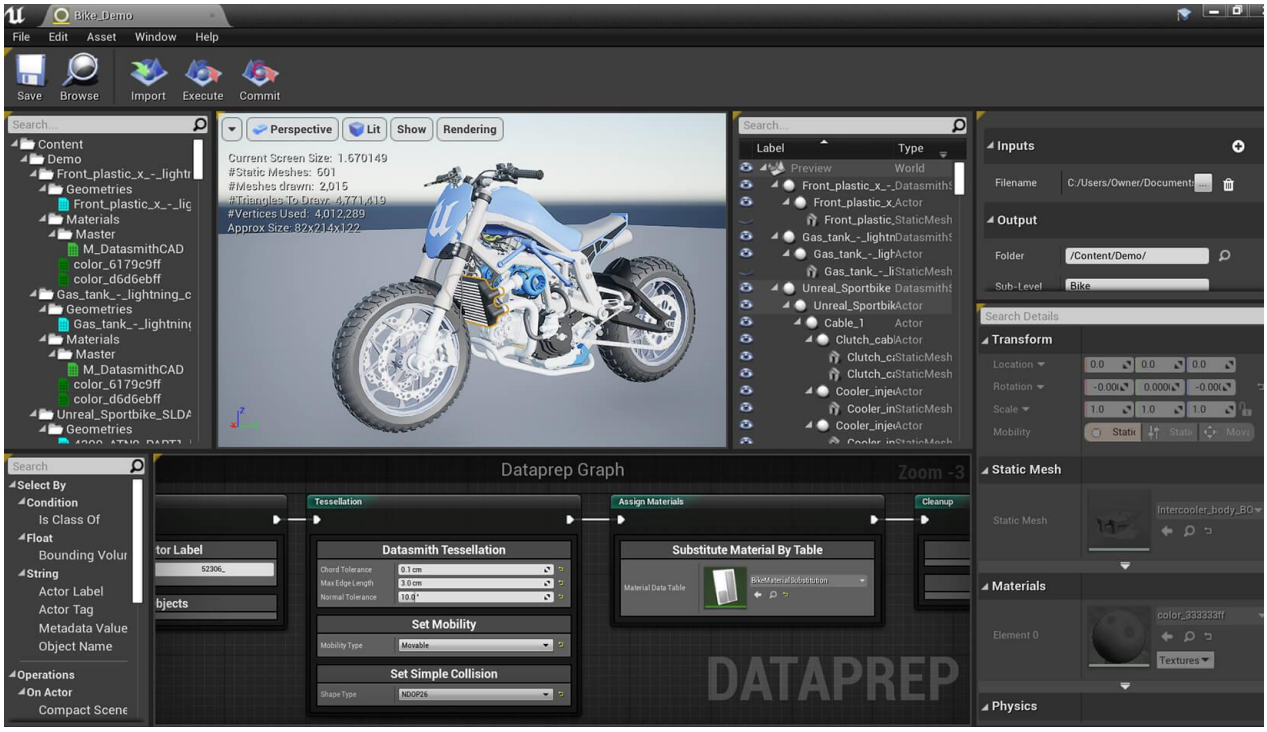


Рис. 2.1 Інтерфейс Unreal Engine

2.1.2 Unity

Unity – це багатоплатформовий ігровий рушій та інструмент для розробки відеоігор розроблений компанією Unity Technologies у 2005 році. Завдяки простоті освоєння є дуже популярним серед розробників початківців, та для розробки інді ігор. Як і Unreal Engine він може використовуватися не лише для створення відеоігор, а й у інших галузях.

Також важливим плюсом Unity є те, що він має безкоштовну версію. Усі ці переваги і роблять його доволі привабливим, як для великих студій, так і для самостійних розробників. Unity має такі функціональні можливості:

- Робота з ресурсами – у Unity інтерфейс складається з багатьох вікон, кожне зі своїм функціоналом і призначенням, які допомагають налагоджувати гру та об’єкти безпосередньо у редакторі.

- Рендеринг - відбувається через віртуальну камеру огляду. Ігрова сцена у робочій області редактору може розміщуватися як завгодно, а при рендерингу — так, як її видно з камери.
- Скрипти – розробники можуть використовувати декілька мов, а саме UnityScript, C# або Boo для написання скриптів, які застосовуються для збільшення функціональності додатку.

За допомогою цього двигуна було розроблено досить багато мобільних ігор, таких як Azur Lane, Call of Duty: Mobile, Fate/Grand Order, також створено і немало комп’ютерних ігор: Enter the Gungeon, Superhot, Raft та багато інших.

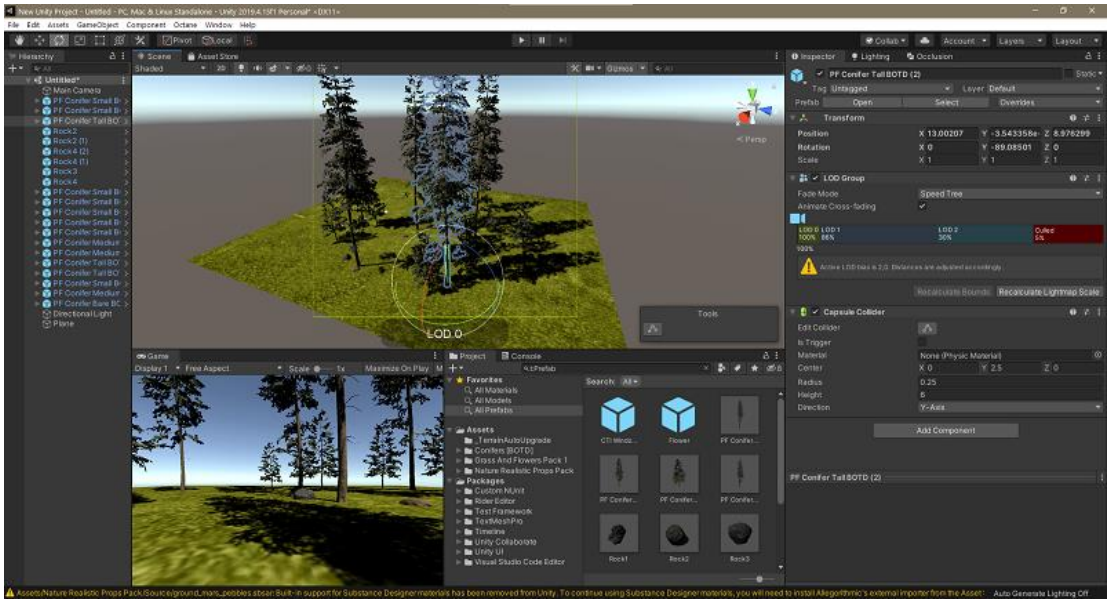


Рис 2.2 Інтерфейс Unity

2.1.3 GameMaker

GameMaker – це кросплатформові ігрові двигуни розроблені YoYo Games. Його особливістю є те, що для створення відеоігор використовується спеціальна мова візуального програмування. Спочатку ця мова була розроблена задля того, щоб

дозволити початківцям створювати відеоігри навіть без особливих знань з програмування, але майбутні версії були також зосереджені на розвитку в сторону просунутих розробників. Його мінусом є те, що безкоштовна версія обмежена компіляцією лише під Windows та декілька обмежений функціонал, але у платній версії стають доступні усі інструменти та компіляція для інших платформ.

Хоч серед ігор створених на рушії GameMaker і не так багато відомих, таких, як Hotline Miami та Super Crate Box, все ж і серед них можливо знайти гру за проходженням якої ви весело проведете свій час.

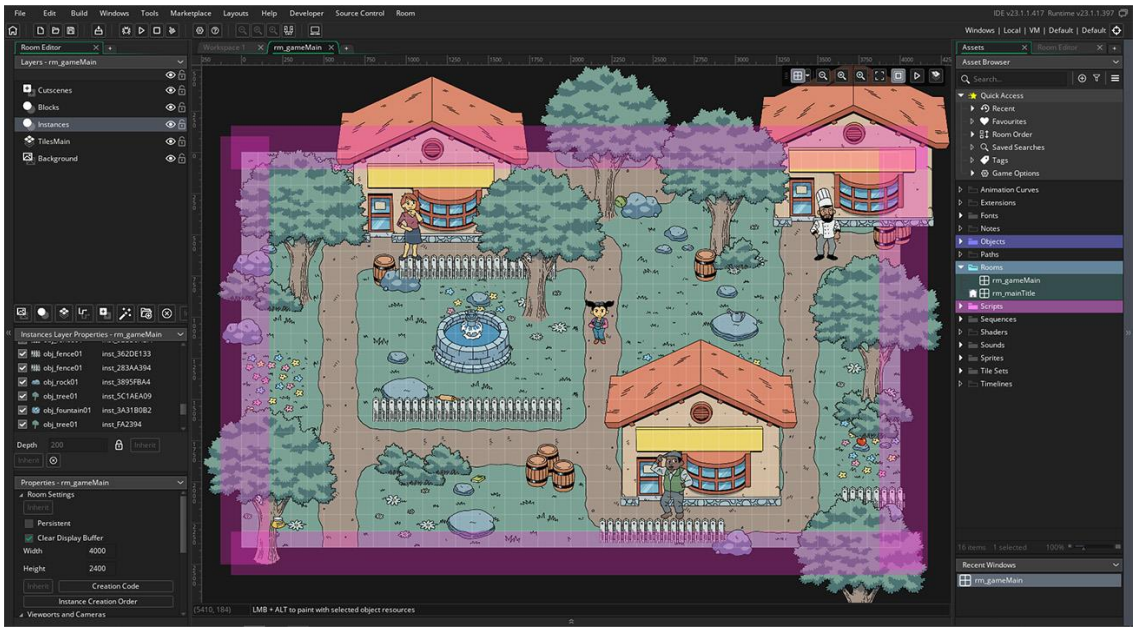


Рис. 2.3 Створення 2D гри за допомогою GameMaker

2.1.4 RPG Maker

RPG Maker – це серія програм для розробки RPG, створених японською групою ASCII. Цей двигун дозволяє створювати ігри навіть без знань з програмування, що робить її ідеальною для початківців, а можливість вносити зміни завдяки JavaScript робить його привабливим і для професійних розробників.

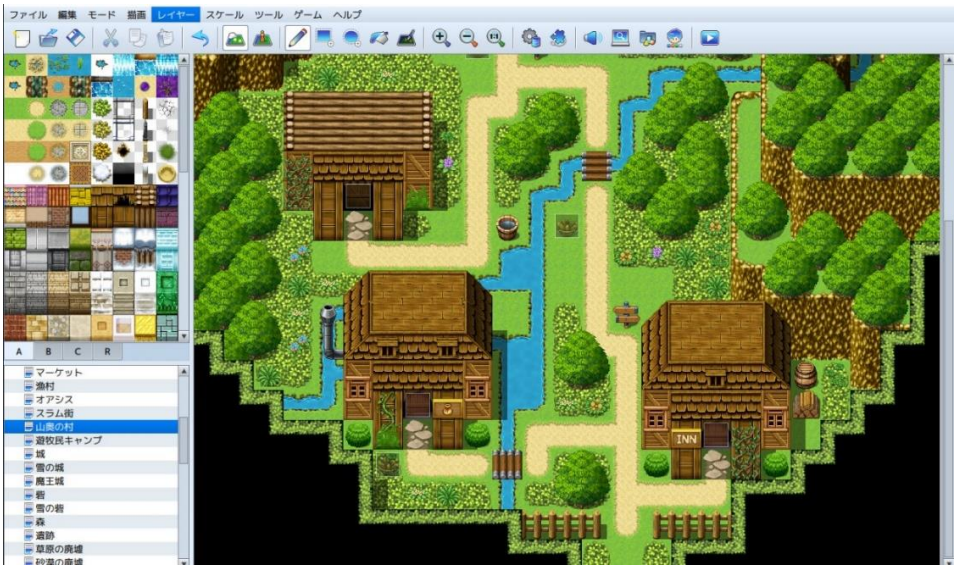


Рис. 2.4 Інтерфейс RPG Maker

2.2 Середовище розробки

Середовищем для розробки було обрано Visual Studio від компанії Microsoft. Це середовище досить популярне серед розробників за його простий інтерфейс(Рис 2.5), підтримку близько 36 різних мов програмування, та за наявність безкоштовної версії. Саме для мого проекту він був обраний через його сумісність з мовою програмування C#, яка у свою чергу чудово підходить для програмування компонентів гри у ігровому рушії Unity.

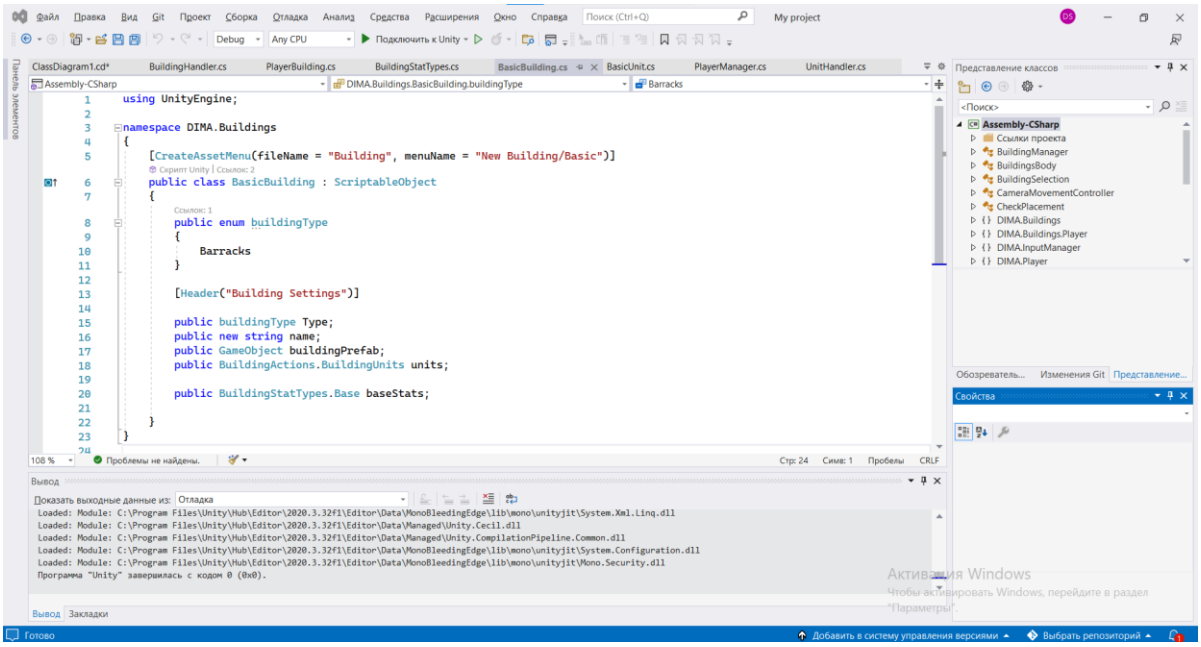


Рис. 2.5 Інтерфейс Visual Studio

2.3 Мова програмування

Мова програмування – це набір правил, який з рядків або графічних елементів програми перетворює їх у різні види виводу машинного коду, і використовуються для реалізації алгоритмів. На сьогоднішній день існує величезна кількість різноманітних мов програмування, найпопулярнішими з них згідно індексу TIOBE(Рис 2.6) являються Python, C, Java, C++, C#.

May 2022	May 2021	Change	Programming Language	Ratings
1	2	▲	Python	12.74%
2	1	▼	C	11.59%
3	3		Java	10.99%
4	4		C++	8.83%
5	5		C#	6.39%
6	6		Visual Basic	5.86%
7	7		JavaScript	2.12%
8	8		Assembly language	1.92%

Рис. 2.6 Індекс TIOBE

2.3.1 Python

На сьогодні Python вважається найпопулярнішою мовою програмування. Він використовується для розробки пакетів зображень та анімації, наукових і обчислювальних програм, розробки відеоігор та багато іншого. Основними його перевагами являються читабельність та велика кількість посібників, що робить її дуже привабливою для початківців, безкоштовна основа та відкритий код. Він являється популярним як у технічній сфері, так і у бізнесі.

2.3.2 C та C++

Мова програмування C, являється однією із найстаріших мов, та є коренем багатьох інших мов, таких як Java, JavaScript, C#, а C++ являється розширеною версією мови C. Різницею між ними являється те, що C більше відноситься до структурного та процедурного програмування, а C++ до об’єктно орієнтованого. Недоліками цих мов програмування являється доволі важке їх опанування.

2.3.3 Java

Також являється однією з найбільш популярних мов програмування, та має доволі високий попит на ринку. Являється об’єктно орієнтованою мовою програмування, а її великою перевагою являється функція Write One, Run Anywhere(WORA), що дозволяє використовувати додатки написані цією мовою незалежно від платформи. Ця мова використовується у розробці програм, веб-розробці, та у big data.

2.3.4 C#

За останній рік рейтинг популярності цієї мови програмування зріс майже на 2%. C# являється однією з найзріліших мов програмування, яка підтримує багато сучасних парадигм. Дана мова програмування являється об’єктно орієнтованою, та найкраще підходить для створення додатків на Windows, Android та iOS. Його перевагами являються швидкість, легкість використання, велика кількість бібліотек та сумісність з іншими кодами.

2.4 Аналіз аудиторії гри

Невід’ємною частиною створення будь-якої гри є аналіз майбутньої аудиторії, це допомагає обрати шлях розвитку майбутнього проекту.

Аудиторією гри було обрано гравців, які тільки вирішили познайомитися з жанром ігор стратегій у реальному часі, та досвід яких у даному жанрі є мінімальним. Для того щоб пригорнути увагу цієї аудиторії було вирішено понизити складність до мінімальної, а швидкість гри зробити такою, щоб у гравців було достатньо часу , щоб обдумати свої майбутні кроки.

2.5 Актуальність проекту

Актуальність проекту вказує на те, чому саме створюється продукт, та для чого його потрібно створювати саме зараз.

Актуальністю даної роботи являється те, що в останні роки популярність жанру почала знижуватися і кількість нових ігор також зменшилася, тому цей проект створюється задля того, щоб пригорнути увагу нових гравців до даного жанру відеоігор, показавши їм переваги у зрозумілому та легкому стилі.

2.6 Функціонал гри

Даний проект повинен включати у себе такий функціонал:

- Виділення юнітів;
- Пересування юнітів;
- Атака ворога або ворогом;
- Будівництво споруд;
- Накопичення та витрата ресурсів;
- Створення юнітів;
- Початок та кінець гри.

2.7 Діаграми

2.7.1 Діаграма класів

Діаграма класів використовується для відображення структури певної системи шляхом моделювання її класів, операцій, атрибутів та зав’язків. Для даного проекту

діаграма класів буде створена за допомогою Visual Studio. Для того, щоб отримати можливість створити діаграму класів потрібно спочатку завантажити додатковий інструмент, а саме Конструктор класів, потім до проекту додається новий елемент під назвою діаграма класів. Для відображення діаграми достатньо перетягнути ваш проект з Solution Explorer до діаграми класів.

Після виконання всіх дій буде створено діаграму класів, яка включатиме усі класи, які знаходяться у даному рішенні(Рис. 2.7)

Рис. 2.7 Діаграма класів

2.7.2 Діаграма варіантів використання

Діаграма варіантів використання – це графічне зображення можливої взаємодії користувача з системою. У ній вказуються різні типи користувачів, та варіанти використання, які має система.

На Рис. 2.8 зображено діаграму варіантів використання для даного проекту. Системою являється сама гра, Actor – це людина, яка грає у цю гру, а варіантами використання являються функції, які гравець може виконувати у грі.

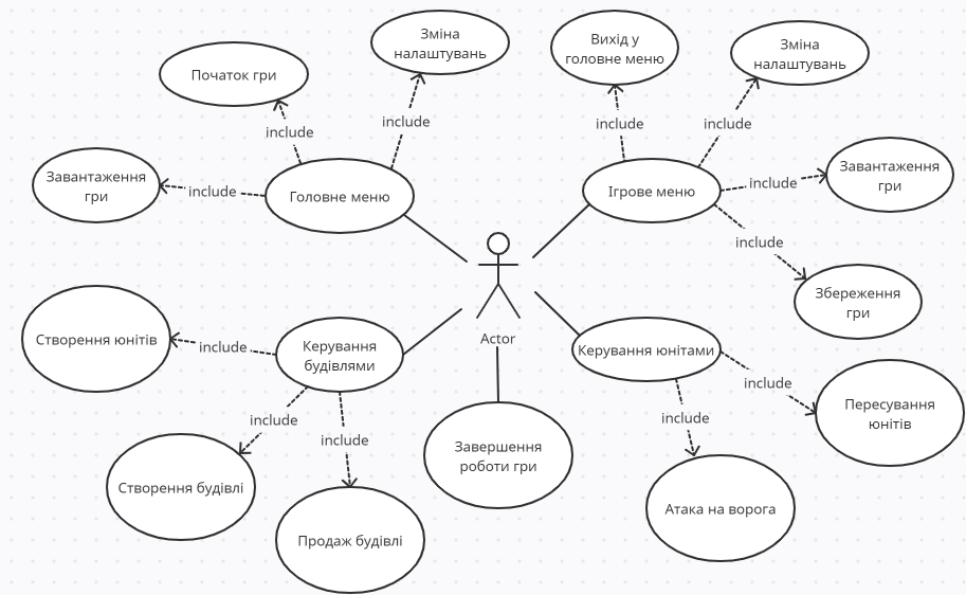


Рис. 2.8 Діаграма варіантів використання

2.8 Висновок до другого розділу

Провівши аналіз існуючих ігрових рушіїв, середовищ розробки та мов програмування для проекту було обрано декілька найбільш підходящих представників. А саме, ігровим рушієм було обрано Unity, середовищем розробки Visual Studio, а мовою програмування C#. Також було проведено аналіз аудиторії гри та визначено актуальність проекту та його функціонал.

3 РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1 Використані програмні інструменти

3.1.1 Paint

Paint – являється простим растровим графічним редактором, розробленим компанією Microsoft. Він входить до усіх версій Windows. Він являється доволі легким у використанні та безкоштовним. У проекті він був використаний для створення фонових зображень та деяких елементів GUI(Рис. 3.1)

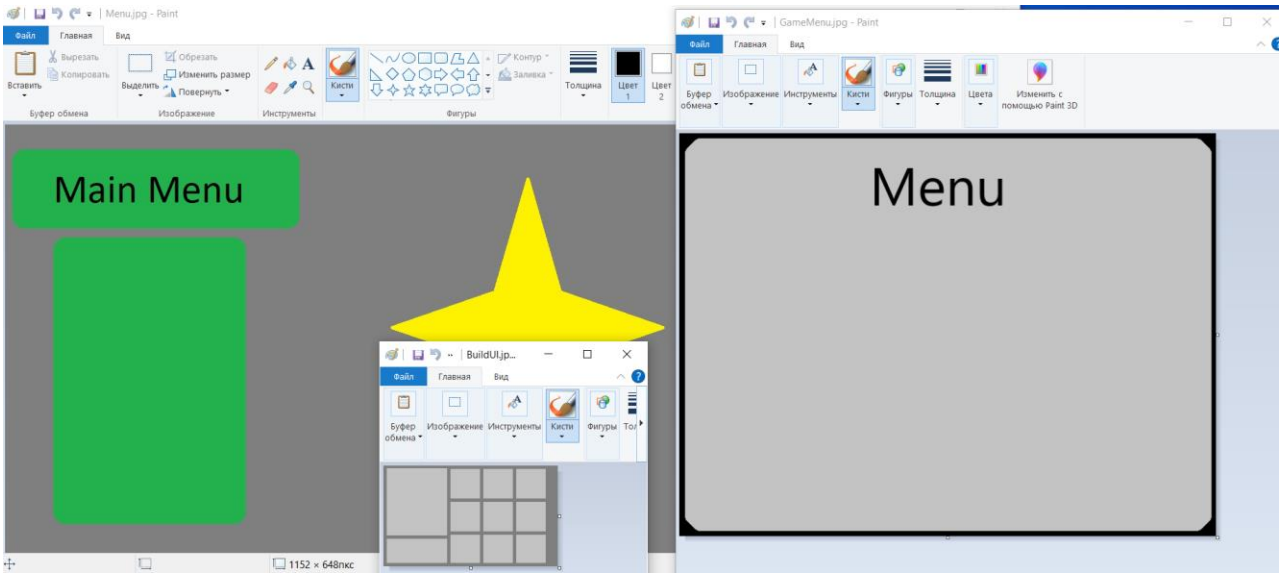


Рис. 3.1 Створення малюнків за допомогою Paint

3.1.2 Blender

Blender – це набір програмних засобів з відкритим вихідним кодом для створення комп’ютерної 3D графіки(Рис. 3.2). Він використовується для створення візуальних ефектів, анімаційних фільмів, анімаційної графіки та іншого. Його особливостями являються моделювання, симуляція, анімація та відтворення. У проекті він використовувався для створення 3D моделей об’єктів.

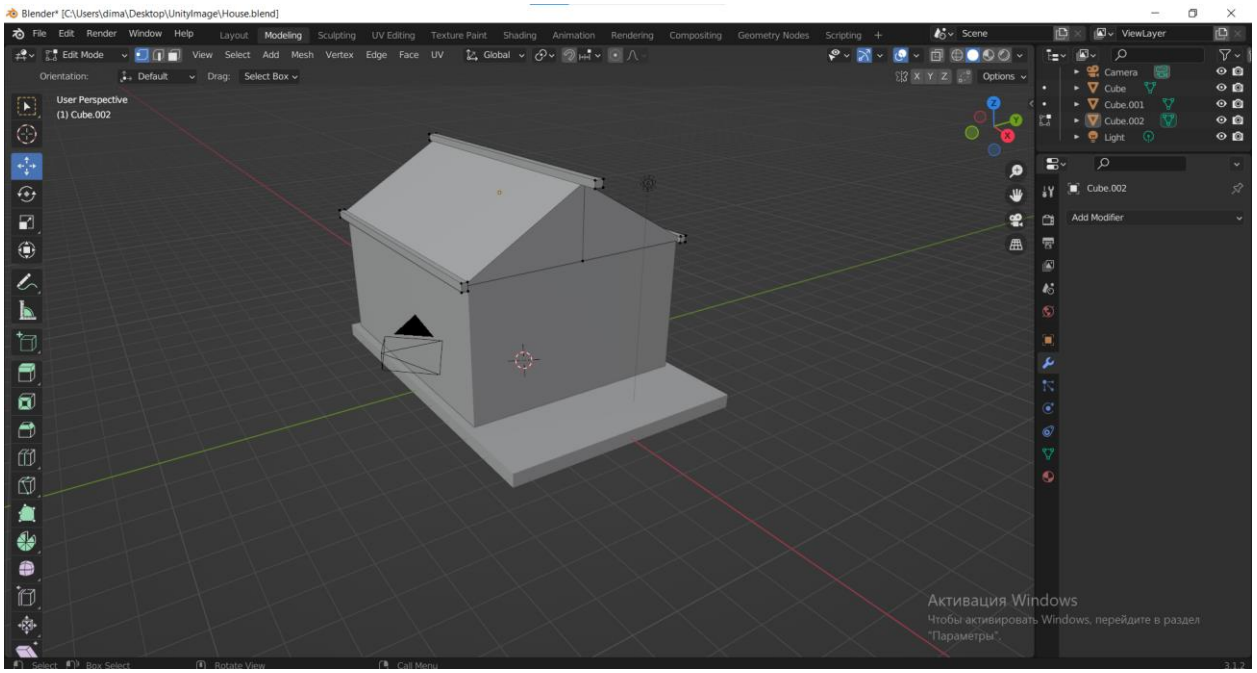


Рис. 3.2 Інтерфейс Blender

3.2 Проектування гри

Після обрання середовища розробки та додаткових програмних інструментів переходимо саме до створення гри.

3.2.1 Головне меню та меню паузи

Перше, що зустрічає гравця після запуску гри – це головне меню(Рис. 3.3). Головне меню включає у себе фон та кнопки для виконання певних дій. Головне меню включає у себе декілька кнопок, а саме:

- Start – розпочинає гру з початку;
- Load – завантажує збережену гру;
- Settings – переключає у меню налаштувань;
- Exit – завершує роботу гри.

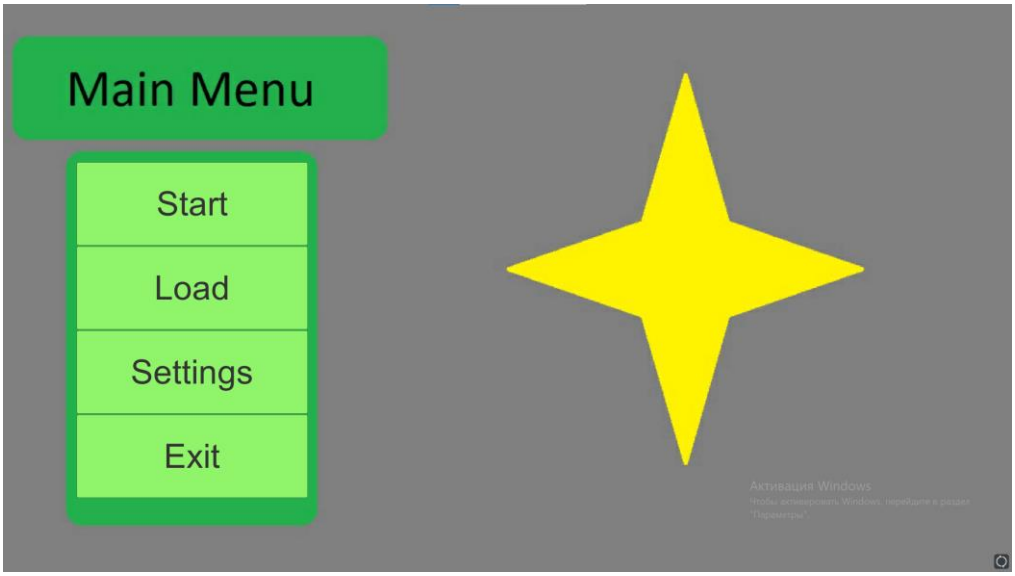


Рис. 3.3 Головне меню гри

Архітектура головного меню показана на Рис. 3.4

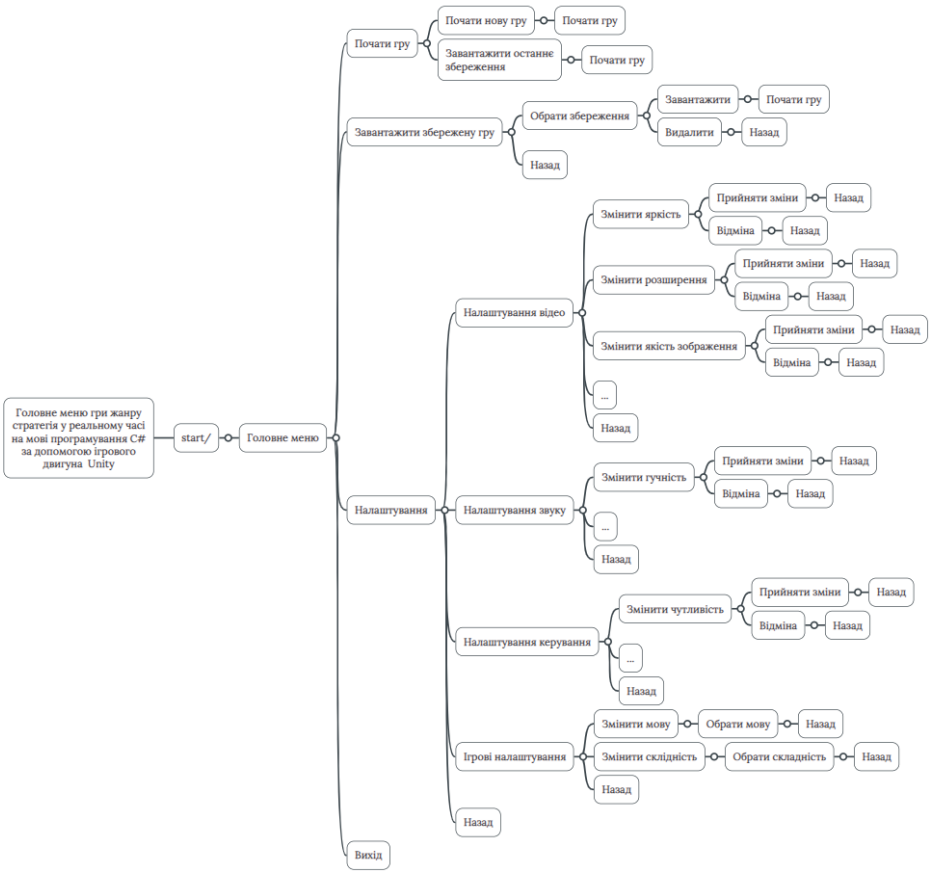


Рис. 3.4 Архітектура головного меню

Ще однією важливою частиною гри є меню паузи(Рис. 3.5). Меню паузи має схожий функціонал до головного меню, але окрім цього включає у себе кнопки зберегти гру(Save) та повернутися до головного меню(Main Menu).

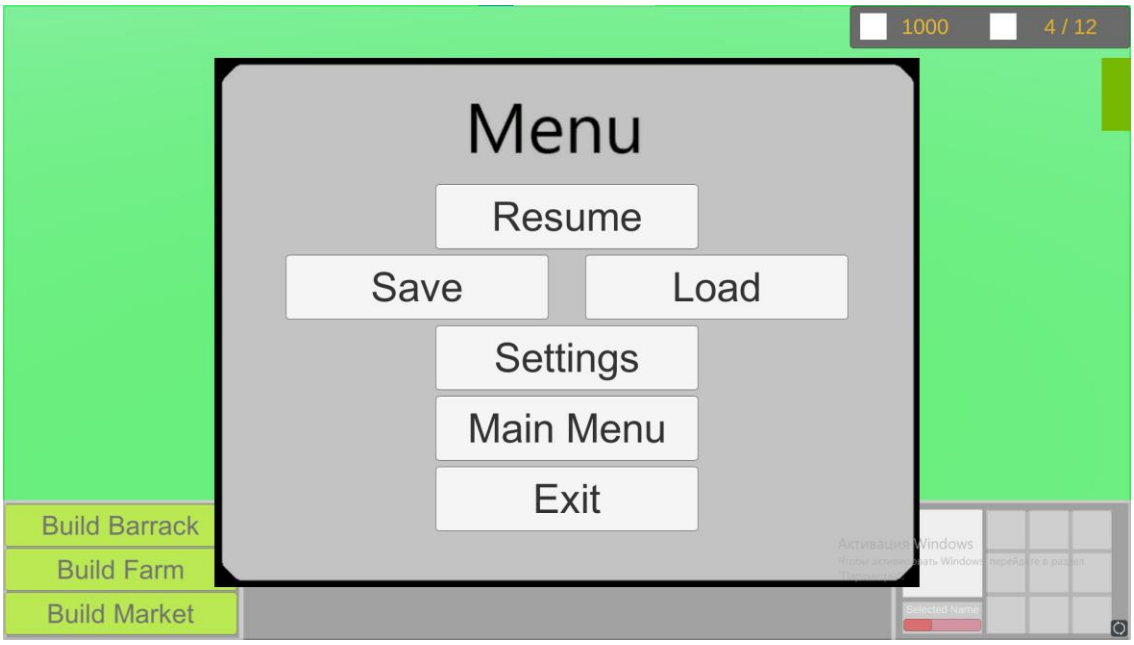


Рис. 3.5 Меню паузи

За роботу головного меню та меню паузи відповідають такі скрипти як GameMenu та MainMenu.

3.2.2 Камера

Камера – це елемент завдяки якому гравець бачить світ гри. Вони являються дуже важливою частиною, та можуть бути змінені, заскриптовані, або прив’язані до якогось елементу, що може додати грі своєї унікальності та шарму. У іграх жанру стратегія у реальному часі камера знаходиться над ігровим полем та може показувати вид вертикально зверху, або під деяким кутом. У даній грі камера знаходиться під кутом 60 градусів до ігрового поля. Вона керується за допомогою клавіш зі стрілками, або клавішами WASD:

```
float hsp = transform.position.y * speed * Input.GetAxis("Horizontal");
        float vsp = transform.position.y * speed * Input.GetAxis("Vertical");
        float scrollSp = Mathf.Log(transform.position.y) * -zoomSpeed * Input.GetAxis("Mouse
ScrollWheel");

        if((transform.position.y >= maxHeight) && (scrollSp > 0))
        {
            scrollSp = 0;
        }
        else if ((transform.position.y <= minHeight) && (scrollSp < 0))
        {
            scrollSp = 0;
        }

        Vector3 verticalMove = new Vector3(0, scrollSp, 0);
        Vector3 lateralMove = hsp * transform.right;
        Vector3 forwardMove = transform.forward;
        forwardMove.y = 0;
        forwardMove.Normalize();
        forwardMove *= vsp;

        Vector3 move = verticalMove + lateralMove + forwardMove;
        Vector3 newPosition = transform.position += move;
              transform.position = Vector3.Lerp(transform.position, newPosition,
Time.deltaTime * 0.5f);
```

Або за допомогою миші:

```
if (Input.mousePosition.x >= Screen.width - 2.25f)
        {
            transform.position += new Vector3(movementSpeed * Time.deltaTime, 0.0f, 0.0f);
        }
        else if (Input.mousePosition.x <= 2.25f)
        {
            transform.position -= new Vector3(movementSpeed * Time.deltaTime, 0.0f, 0.0f);
        }
        if (Input.mousePosition.y >= Screen.height - 2.25f)
        {
            transform.position += new Vector3(0.0f, 0.0f, movementSpeed * Time.deltaTime);
        }
        else if (Input.mousePosition.y <= 2.25f)
        {
            transform.position -= new Vector3(0.0f, 0.0f, movementSpeed * Time.deltaTime);
          }
```

А швидкість її руху визначається глобальною змінною movementSpeed.

3.2.3 Юніти

Юніти є однією з найважливіших частин ігор жанру RTS, це ігрові об’єкти, які можуть пересуватися та використовуються для деяких цілей, таких як атака ворога, видобування ресурсів, ремонт або зцілення інших юнітів або споруд. Вони можуть належати як гравцеві, так і бути нейтральними або ворожими до нього.

У кожного юніта присутній набір базових характеристик, які присвоюються їм після початку гри, або їх безпосереднім створення(купівлею, найняттям). Для того, щоб присвоїти юнітам їх характеристики, було створено декілька класів, а саме: клас UnitStatTypes, у якому зберігаються такі характеристики як:

- Атака – вказує скільки здоров'я забере юніт у ворога при проведенні атаки.
- Дистанція атаки – це дистанція у межах якої юніт може атакувати ворога
- Швидкість атаки – це час до наступної атаки.
- Здоров'я – це основна характеристика, яка показує цілісність персонажа, коли вона опускається до 0 юніт помирає.
- Мана – характеристика присутня не у всіх юнітів, зменшується при використанні спеціальних навичок.
- Захист – показує на скільки забереться одиниць здоров'я при отриманні атаки від ворога.
- Дистанція агресії – величина, яка показує наскільки далеко має бути ворог, щоб даний юніт напав на нього без команди гравця.

Наступним буде клас BasicUnit, який являється ScriptableObject. ScriptableObject – це код класу, що дозволяє створювати в Unity Scriptable Objects для зберігання великих об'ємів даних, не залежних від екземплярів скриптів.

```
public class BasicUnit : ScriptableObject
    {
        public enum UnitTypes
        {
            Worker,
            Swordsman,
            Archer,
            Healer,
            Defender
        }

        [Header("Unit Settings")]

        public UnitTypes type;
        public new string name;
        public GameObject playerUnitPrefab;
        public GameObject enemyUnitPrefab;
```

```
    public GameObject icon;
    public float spawnTime;
    public float goldCost;
    public float foodCost;

    public UnitStatTypes.Base baseStats;
      }
```

Цей клас зберігає у собі:

- Тип юнітів – вказує до якого типу відноситься даний юніт.
- Ім’я – назва юніта.
- Префаб для юнітів гравця та ворога – Prefab – це ассет, який являється шаблоном якого GameObject. Отже префаб для юніта гравця або ворога – це об’єкт який і буде відображатися на ігровому полі та мати у собі основні скрипти.
- Іконка – це картинка, яка відображується на кнопках найму юніта, або при його виділенні.
- Час створення – це час за який юніт буде створений на ігровому полі після того, як його почнуть наймати.
- Ціна – це кількість ресурсів, яка буде витрачена на найм юніта.
- Їжа – це кількість місця, яке буде займати даний юніт. Під місцем вважається деяке число, яке визначає кількість юнітів доступну для розміщення.

Наступним завданням буде присвоїти значення юнітам у грі, за це відповідають скрипти UnitHandler, у кому знаходиться функція, яка по переданому типу юніта обирає які саме характеристики йому присвоїти:

```
private BasicUnit worker, swordsman, archer;

public UnitStatTypes.Base GetBasicUnitStats(string type)
        {
            BasicUnit unit;

            switch (type)
            {
                case "worker":
                    unit = worker;
```

```
                break;
            case "swordsman":
                unit = swordsman;
                break;
            case "archer":
                unit = archer;
                break;
            default:
                return null;
        }
        return unit.baseStats;
          }
```

Та клас PlayerManager, який саме і обирає якому об’єкту які значення будуть присвоєні:

```
public void SetBasicStats(Transform type)
        {
            foreach (Transform child in type)
            {
                foreach (Transform tf in child)
                {
                    string name = child.name.Substring(0, child.name.Length - 1).ToLower();
                    if (type == playerUnits)
                    {
                        Units.Player.PlayerUnit pUnit =
tf.GetComponent<Units.Player.PlayerUnit>();
                        pUnit.baseStats =
Units.UnitHandler.instance.GetBasicUnitStats(name);
                    }
                    else if (type == enemyUnits)
                    {
                        Units.Enemy.EnemyUnit eUnit =
tf.GetComponent<Units.Enemy.EnemyUnit>();
                        eUnit.baseStats =
Units.UnitHandler.instance.GetBasicUnitStats(name); ;
                    }
                    else if (type == playerBuildings)
                    {
                        Buildings.Player.PlayerBuilding pBuilding =
tf.GetComponent<Buildings.Player.PlayerBuilding>();
                        pBuilding.baseStats =
Buildings.BuildingHandler.instance.GetBasicBuildingStats(name);
                    }
                }
            }
          }
```

Далі потрібно якось відобразити характеристики персонажів у самій грі. Для цього було створено скрипт UnitStatDisplay, який відповідає за відображення полоси здоров’я персонажів та споруд, а також її зміну та знищення персонажів. Для початку було створене префаб UnitStats(Рис. 3.6), який включав у себе два UI

елементи image, які відповідали за полосу здоров’я юнітів та до якого було підключено скрипт UnitStatDisplay.

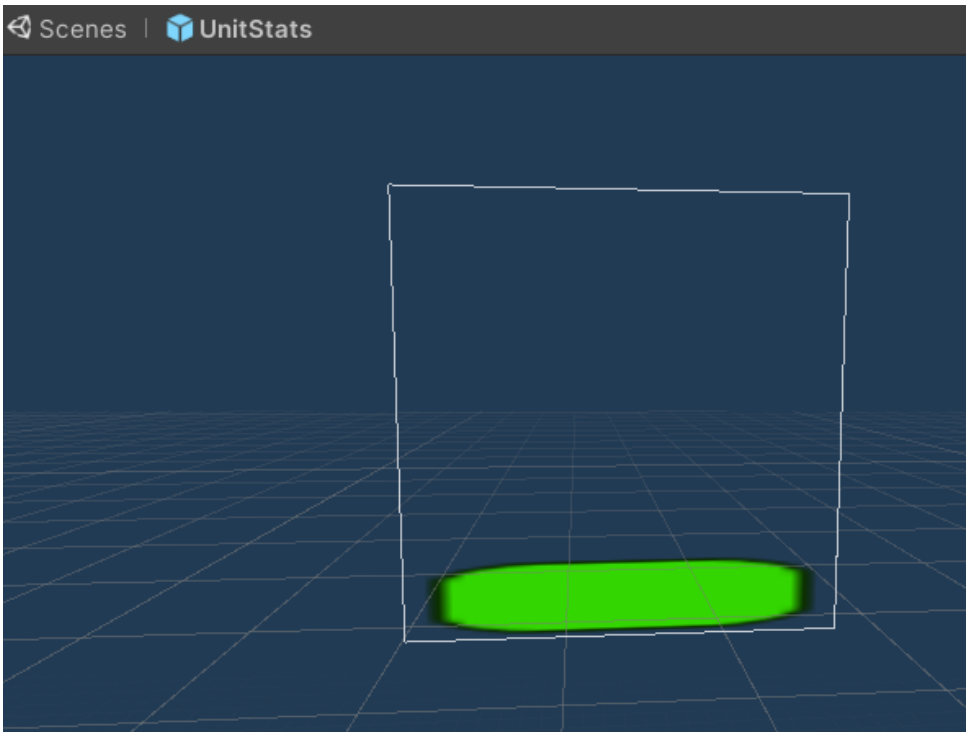


Рис. 3.6 Префаб UnitStats

Далі у скрипті UnitStatDisplay було додано функції, які відповідали за такі завдання, як присвоєння початкових значень:

```
public void SetStatDisplayBasicUnit(UnitStatTypes.Base stats,bool isPlayer)
        {
            maxHP = stats.HP;
            armor = stats.armor;
            isPlayerUnit = isPlayer;

            currHP = maxHP;
              }
```

Отримання урону:

```
public void TakeDamage(float b)
        {
            Debug.Log(b);
            float totalDamage = b - armor;
```

```
            currHP -= totalDamage;
              }
```

Правильне відображення на екрані:

```
private void HandleHeath()
        {
            Camera cam = Camera.main;
            gameObject.transform.LookAt(gameObject.transform.position +
                cam.transform.rotation * Vector3.forward, cam.transform.rotation *
Vector3.up);

            hpBarAmount.fillAmount = currHP / maxHP;

            if(currHP <= 0)
            {
                Die();
            }
              }
```

Та знищення юніта:

```
private void Die()
        {
            if (isPlayerUnit)
            {

InputManager.InputHandler.instance.selectedUnits.Remove(gameObject.transform.parent);
                Destroy(gameObject.transform.parent.gameObject);
            }
            else
            {
                Destroy(gameObject.transform.parent.gameObject);
            }

              }
```

Та на кінець було створено основні скрипти юнітів, а саме PlayerUnit та EnemyUnit для персонажів гравця та ворога відповідно. Ці скрипти відповідали за виклик функцій із інших скриптів, пересування персонажів, перевірку на наявність ворогів у дистанції агресії та за рух до ворога, якщо він помічений.

Приклад функції, яка відповідає за перевірку на наявність ворога у зоні досягнення:

```
private void CheckForEnemy()
        {
            rangeColliders = Physics.OverlapSphere(transform.position, baseStats.aggroRange,
UnitHandler.instance.pUnitLayer);

            for (int i = 0; i < rangeColliders.Length;)
            {
```

```
            aggrTarget = rangeColliders[i].gameObject.transform;
            aggrUnit = aggrTarget.gameObject.GetComponentInChildren<UnitStatDisplay>();
            if (aggrUnit)
            {
                isAggr = true;
            }
            break;
        }
    }
```

3.2.4 Споруди

Споруди – це статичні об’єкти, які можуть виконувати деякі функції, від найму юнітів та збору ресурсів, до атаки ворога. В основному скрипти які відповідають за присвоєння характеристик спорудам дуже схожі на такі для юнітів, різницею є кількість характеристик. Основні характеристики, які присвоюються спорудам такі: здоров’я, захист та атака(якщо потрібно).

3.2.5 Ресурси

Ігрові ресурси також вважаються важливою частиною ігор жанру RTS, у даній грі ресурсів є два типи, це золото та їжа. Золото – це ресурс який видобувається за допомогою споруди Market, і використовується для створення нових юнітів або будівництва споруд. Їжа – це ресурс який відповідає за кількість доступних для найму персонажів, цей ресурс має своє максимально доступне, доступне та використане значення(Рис. 3.7).

Рис. 3.7 Вигляд панелі ресурсів

3.2.6 Керування грою

Керування включає у себе такі функції як, виділення юнітів, керування юнітами, виділення будівель, будівництво будівель, керування будівлями.

Такі скрипти як InputHandler та MultiSelect відповідають за виділення споруд або юнітів для подальшого керування ними. Функція HandleUnitMovement відповідає за виділення споруди, або юніта за допомогою лівої клавіші миші, та за рух юнітів за допомогою правої клавіші миші, при цьому відбувається перевірка чи є хоча б один виділений юніт. Функція DeselectUnit спрацьовує тоді, коли гравець натискає ліву клавішу миші при цьому не навівши курсор на ігрового юніта чи будівлю, які належать йому, та відміняє їх виділення. Функції AddedUnit та AddedBuilding відповідають за додавання обраних об’єктів до списку.

```
if (iUnit)
            {
                if (!canMultiselect)
                {
                    DeselectUnits();
                }

                selectedUnits.Add(iUnit.gameObject.transform);

                iUnit.OnInteractEnter();

                return iUnit;
            }
            else
            {
                return null;
                  }
```

Клас MultiSelect відповідає за малювання спеціального прямокутника(Рис. 3.8), який вказує область на якій гравець хоче виділити юнітів.

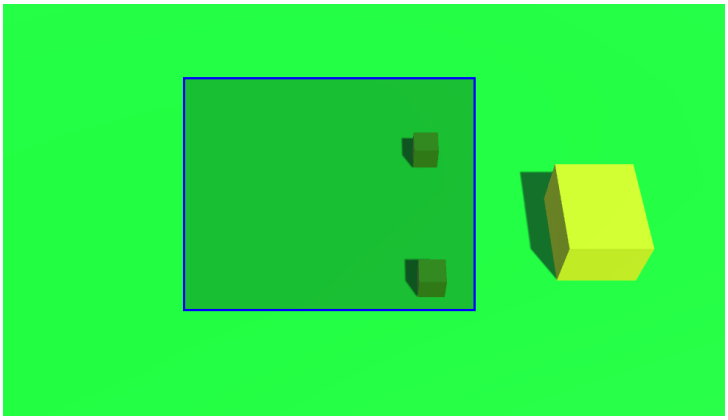


Рис. 3.8 Область виділення

Він викликається у класі InputHandler у функції OnGUI:

```
private void OnGUI()
        {
            if (isDragging)
            {
                Rect rect = MultiSelect.GetScreenRect(mousePosition, Input.mousePosition);
                MultiSelect.DrawScreenRect(rect, new Color(0f, 0f, 0f, 0.25f));
                MultiSelect.DrawScreenRectBorder(rect, 3,Color.blue);
            }
              }
```

А сам вибір юнітів відбувається у функції IsWhithinSelectionBounds:

```
private bool IsWhithinSelectionBounds(Transform obj)
        {
            if (!isDragging)
            {
                return false;
            }

            Camera cam = Camera.main;
            Bounds vpBounds = MultiSelect.GetVPBounds(cam, mousePosition,
Input.mousePosition);
            return vpBounds.Contains(cam.WorldToViewportPoint(obj.position));
              }
```

Також для виділення використовуються скрипти Interactable, IBuildings, IUnits, які відповідають за підсвічення виділених юнітів та будинків, а скрипт IBuildings також відповідає за відображення позиції(Рис. 3.9) у яку підуть юніти після найму.

Рис. 3.9 Точка збору юнітів

Будівництво будівель відбувається за допомогою двох скриптів, а саме BuildingManager та CheckPlacement, та кнопок на головному екрані(Рис. 3.10).

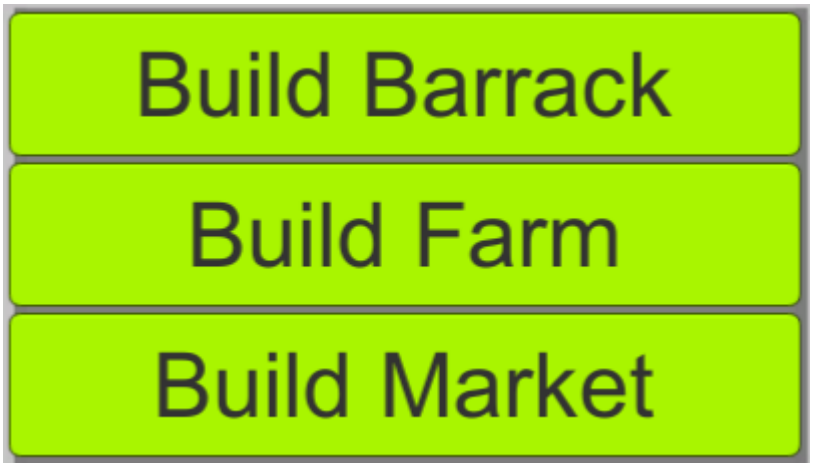


Рис. 3.10 Кнопки для будівництва споруд

При натисненні однієї з кнопок якщо гравцеві буде достатньо ресурсів, буде обрано споруду для подальшого її будівництва. Після цього на екрані буде відображено силует, який слідкуватиме за курсором миші гравця і показуватиме чи можливо побудувати споруду у тому чи іншому місці. Якщо створенню споруди

нічого не заважатиме, то колір силует матиме зелений колір(Рис. 3.11), а якщо на місці побудови знаходитиметься якийсь інший об’єкт то він змінить свій колір на червоний(Рис. 3.12). Якщо гравець захоче відмінити спробу будівництва споруди, то він може натиснути праву клавішу миші, або кнопку ESC.

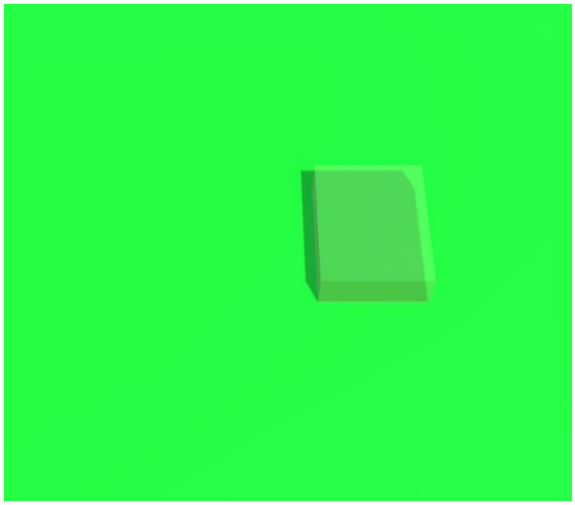


Рис. 3.11 Спроба будівництва без перешкод

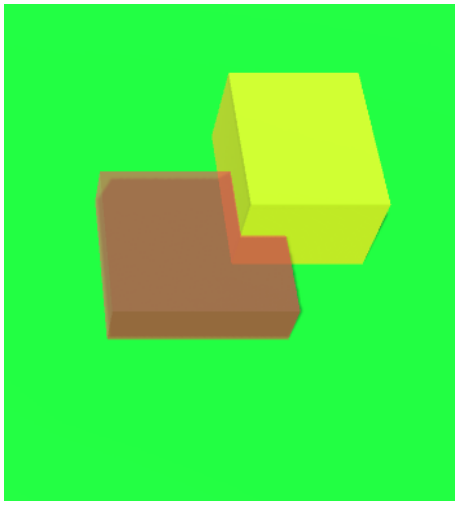


Рис. 3.12 Спроба будівництва з наявною перешкодою

Після створення споруди гравець може виділити її, за це відповідає той самий скрипт, що відповідає і за виділення юнітів. При виділенні споруди у правому нижньому куті екрану відобразиться панель, яка показуватиме деяку інформацію про споруду та включатиме у себе кнопки з діями(Рис. 3.13). На панелі знаходяться такі елементи:

- Назва споруди;
- Здоров’я;
- Іконка;
- Кнопки з діями.

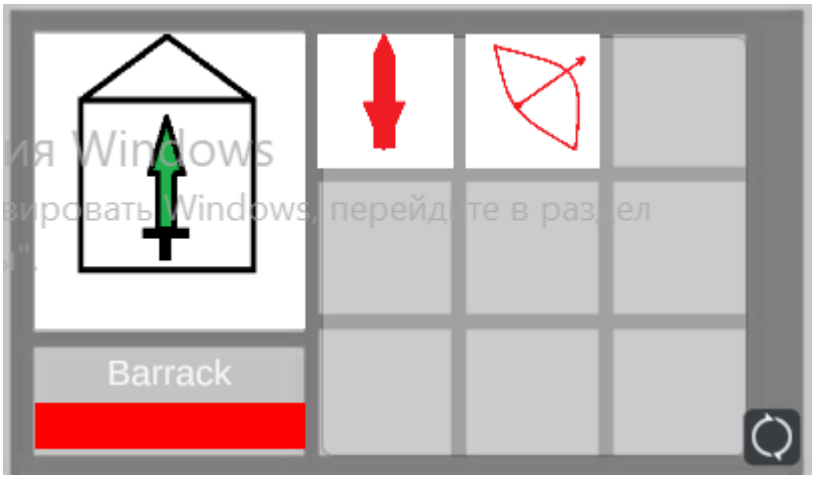


Рис. 3.13 Панель керування спорудою

За найм спорудою юнітів відповідають скрипти Action, ActionFrame та ActionTimer. Скрипт Action має у собі функцію OnClick(), яка під’єднується до кнопок, які відповідають за створення юнітів, та запускає StartSpawnTimer, яка знаходиться у класі ActionFrame, та яка відповідає за запуск таймеру до появи нового юніта, генерацію черги побудови юнітів та за появу юнітів на ігровому полі.

```
public void OnClick()
        {
```

```
            ActionFrame.instance.StartSpawnTimer(name);
        }

public void StartSpawnTimer(string objectToSpawn)
        {
            if (IsUnit(objectToSpawn))
            {
                Units.BasicUnit unit = IsUnit(objectToSpawn);
                spawnQueue.Add(unit.spawnTime);
                spawnOrder.Add(unit.playerUnitPrefab);
            }

            else
            {
                Debug.Log($"{objectToSpawn} is not a spawnable object");
            }

            if (spawnQueue.Count == 1)
            {
                ActionTimer.instance.StartCoroutine(ActionTimer.instance.SpawnQueueTimer());
            }
            else if (spawnQueue.Count == 0)
            {
                ActionTimer.instance.StopAllCoroutines();
            }
        }
```

Також після виділення споруди буде видно «прапорець», який вказуватиме куди буде рухатися юніт після створення(синій об’єкт на Рис. 3.14)

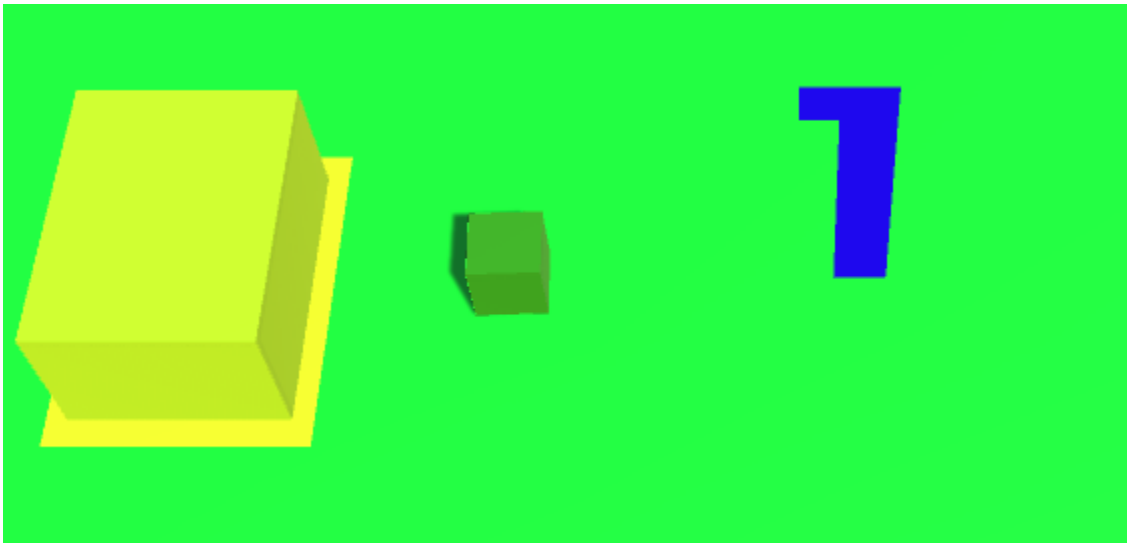


Рис. 3.14 Точка збору юнітів

ВИСНОВКИ

У рамках виконання цієї дипломної роботи було здійснено за допомогою ігрового двигуна Unity розробку гри жанру стратегія у реальному часі для комп’ютерів.

Проаналізовано жанр ігор стратегія у реальному часі та деяких ігор представників цього жанру. Виявлено переваги та недоліки кожної гри. Також було проведено аналіз засобів розробки, які використовувалися для виконання даної роботи, також було виявлено їх особливості, які вирізняють їх серед альтернативних варіантів.

Визначено алгоритм розробки гри, її потенційна аудиторія, актуальність та необхідний функціонал. Створено та розібрано деякі діаграми.

У наступному етапі було ретельно розібрано функціонал гри, створені методи та класи, об’єкти.

Результатом виконання даної кваліфікаційної роботи являється реалізована гра жанру стратегія у реальному часі, яка включає у себе функціонал класичних RTS, та яка у майбутньому може використовуватися як шаблон для створення нових ігор подібного жанру.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Cossacks Wiki [Електронний ресурс]. Режим доступу до ресурсу: https://cossacks.fandom.com/wiki/Cossacks:_Back_to_War
2. What Is A Real-Time Strategy Game? An Exploration and Definition [Електронний ресурс]. Режим доступу до ресурсу: https://waywardstrategy.com/2015/09/25/what-is-an-rts-game/
3. The Fall and Rise of Real-Time Strategy Games [Електронний ресурс]. Режим доступу до ресурсу: https://www.wired.com/story/fall-and-rise-real-time-strategy-games/
4. Attractiveness of Real Time Strategy Games [Електронний ресурс]. Режим доступу до ресурсу: https://www.researchgate.net/publication/281799942_Attractiveness_of_Real_Time_Strategy_Games
5. Blender [Електронний ресурс]. Режим доступу до ресурсу: https://www.blender.org/about/
6. Unity [Електронний ресурс]. Режим доступу до ресурсу: https://unity.com/solutions/game
7. The Many Different Types of Video Games & Their Subgenres [Електронний ресурс]. Режим доступу до ресурсу: https://www.idtech.com/blog/different-types-of-video-game-genres
8. Video Game Genres: Everything You Need to Know [Електронний ресурс]. Режим доступу до ресурсу: https://www.hp.com/us-en/shop/tech-takes/video-game-genres
9. WHAT ARE THE BEST PLATFORMS FOR VIDEO GAMES? [Електронний ресурс]. Режим доступу до ресурсу: https://starloopstudios.com/what-are-the-best-platforms-for-video-games/
10. A GUIDE TO THE DIFFERENT TYPES OF VIDEO GAME PLATFORMS

[Електронний ресурс]. Режим доступу до ресурсу: https://glassyeyewear.com/blogs/article/a-guide-to-the-different-types-of-video-game-platforms

11. The Complete Guide to Video Game Genres: From Scrollers, Shooters, to Sports [Електронний ресурс]. Режим доступу до ресурсу: https://www.gamedesigning.org/gaming/video-game-genres/
12. Unreal Engine [Електронний ресурс]. Режим доступу до ресурсу: https://www.unrealengine.com/en-US
13. 10 найкращих ігрових рушіїв [Електронний ресурс]. Режим доступу до ресурсу: https://ulab.sumdu.edu.ua/uk/10-najkrashhih-igrovih-rushiiv
14. TIOBE Index [Електронний ресурс]. Режим доступу до ресурсу: https://www.tiobe.com/tiobe-index/
15. Best Programming Languages to Learn in 2022 [Електронний ресурс]. Режим доступу до ресурсу: https://www.simplilearn.com/best-programming-languages-start-learning-today-article
16. Навчайтеся, навчайте та сертифікуйте з Unity [Електронний ресурс]. Режим доступу до ресурсу: https://unity.com/learn
17. Michael Moore. Basics of Game Design. CRC Press, 2016. 400 с.
18. Wolf, Mark J. P.The Video Game Explosion: A History from PONG to Playstation and Beyond / Mark J. P. Wolf. ABC-CLIO, 2008. — 380 с.

ДОДАТОК А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка гри жанру стратегія у реальному часі на мові програмування C# за допомогою ігрового двигуна Unity

Виконав студент 4 курсу

Групи ПД-44

Сокирко Дмитро Олександрович

Керівник роботи

Доктор філософії(PhD), ПЗ Дібрівний Олесь Андрійович

Київ - 2022

Аналоги

Cossacks Back to War

Warcraft 3 Reign of Chaos

ПОРІВНЯННЯ АНАЛОГІВ

Назва	Переваги	Недоліки
Cossacks Back to War	Різноманітність ігрових сторін	Відсутність підтримки сучасного обладнання
	Реалістична фізика	Відсутні підтримки сучасного програмного забезпечення
	Відсутність обмеження кількості юнітів(ігрових персонажів)	
Warcraft 3 Reign of Chaos	Цікавий сюжет	Застарівша графіка
	Велика кількість карт	Незавершений сюжет
	Наявність редактору карт	

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – привернути увагу нових гравців до ігор жанру стратегія у реальному часі;

Об’єкт дослідження – ігровий процес жанру стратегія у реальному часі;

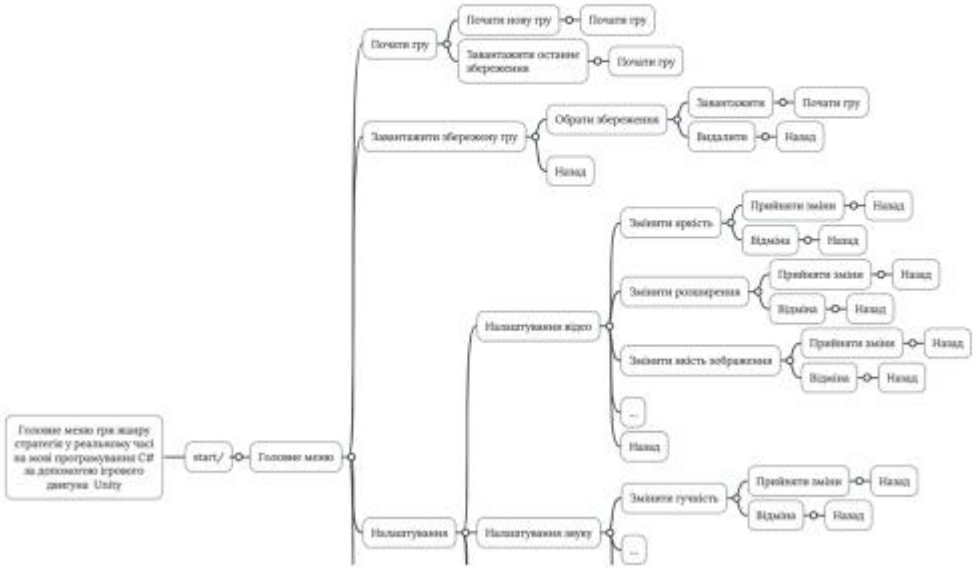
Предмет дослідження – програмні засоби для розробки гри жанру стратегія у реальному часі.

ТЕХНІЧНЕ ЗАВДАННЯ

1. Розробити архітектуру комп'ютерної гри
2. Реалізувати головне меню
3. Реалізувати меню налаштувань
4. Реалізувати створення ігрових юнітів та споруд
5. Реалізувати систему керування юнітами
6. Реалізувати систему збереження та завантаження сцени

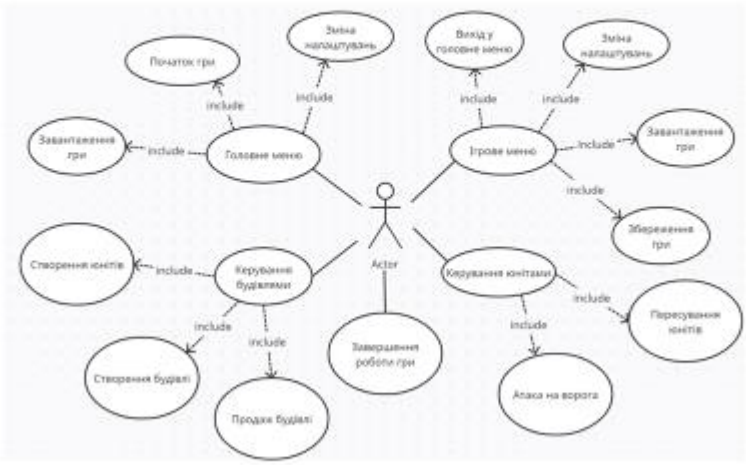
ПРОГРАМНІ ТА ТЕХНІЧНІ ЗАСОБИ РЕАЛІЗАЦІЇ

АРХІТЕКТУРА ГОЛОВНОГО МЕНЮ

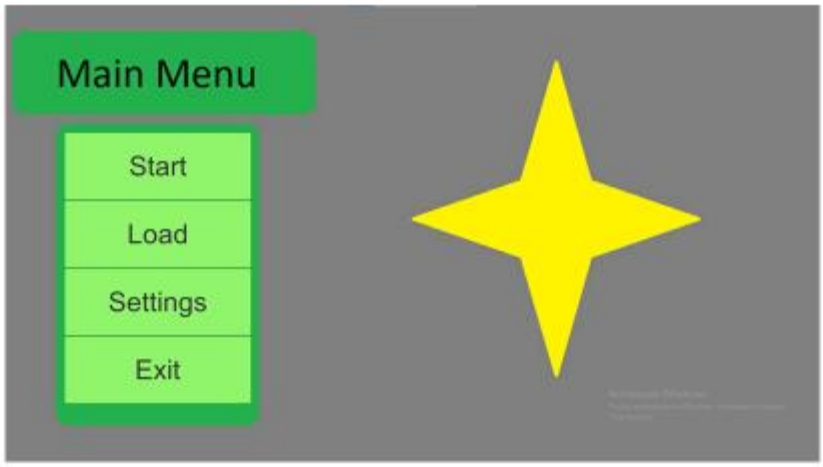


АРХІТЕКТУРА ГОЛОВНОГО МЕНЮ(ПРОДОВЖЕННЯ)

Діаграма прецедентів

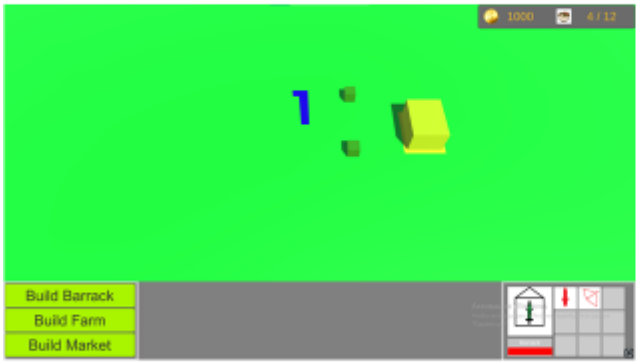


Екранні форми



Головне меню гри

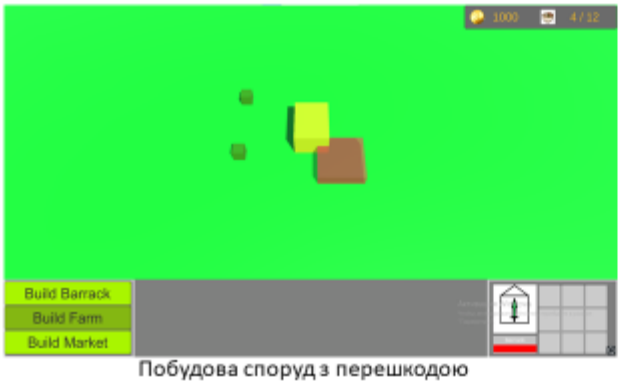
Екранні форми



Ігрове поле

Меню паузи

Екранні форми



Побудова споруд з перешкодою

Побудова споруд без перешкоди

Апробація результатів дослідження

Сокирко Д.О., Штучний інтелект у відеоіграх /Науково-технічна конференція «Застосування програмного забезпечення в інфокомунікаційних технологіях». Збірник тез 20.04.2022, ДУТ, м.Київ – К.: ДУТ, 2022. С. 58-59

ВИСНОВОК

1. Проведено аналіз декількох популярних ігор представників жанру стратегії у реальному часі, виявлено їх переваги та недоліки.
2. Розроблено діаграму прецедентів та схему архітектури головного меню.
3. За допомогою ігрового рушія Unity, та середовища програмування Visual Studio було створено такі основні компоненти гри як:
    1. Можливість почати нову гру, чи продовжити зі збереження;
    2. Можливість зміни основних налаштувань;
    3. Можливість керування юнітами та спорудами, їх створенням, знищенням, та діями;
    4. Можливість керування ігровими ресурсами, їх видобування та реалізація.

ДЯКУЮ ЗА УВАГУ!

ДОДАТОК Б

Деякі з скриптів:

```
using UnityEngine;

namespace DIMA.Units
{
    [CreateAssetMenu(fileName = "New Unit", menuName = "New Unit/Basic")]
    public class BasicUnit : ScriptableObject
    {
        public enum UnitTypes
        {
            Worker,
            Swordsman,
            Archer,
            Healer,
            Defender
        }

        [Header("Unit Settings")]

        public UnitTypes type;
        public new string name;
        public GameObject playerUnitPrefab;
        public GameObject enemyUnitPrefab;
        public GameObject icon;
        public float spawnTime;
        public float goldCost;
        public float foodCost;

        public UnitStatTypes.Base baseStats;
    }
}


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

namespace DIMA.Units
{
    public class UnitStatTypes : ScriptableObject
    {
        [System.Serializable]
        public class Base
        {
            public float attack, attackRange,atkSpeed, HP, MP, armor, aggroRange, speed;
        }
    }
}


using UnityEngine;
using UnityEngine.UI;

namespace DIMA.Units
{
    public class UnitStatDisplay : MonoBehaviour
    {
```

```
        public float maxHP, armor, currHP;

        [SerializeField] private Image hpBarAmount;

        private bool isPlayerUnit = false;

        public void SetStatDisplayBasicUnit(UnitStatTypes.Base stats,bool isPlayer)
        {
            maxHP = stats.HP;
            armor = stats.armor;
            isPlayerUnit = isPlayer;

            currHP = maxHP;
        }

        public void SetStatDisplayBasicBuilding(Buildings.BuildingStatTypes.Base stats, bool
isPlayer)
        {
            maxHP = stats.HP;
            armor = stats.armor;
            isPlayerUnit = isPlayer;

            currHP = maxHP;
        }

        void Update()
        {
            HandleHeath();
        }

        public void TakeDamage(float b)
        {
            Debug.Log(b);
            float totalDamage = b - armor;
            currHP -= totalDamage;
        }
        private void HandleHeath()
        {
            Camera cam = Camera.main;
            gameObject.transform.LookAt(gameObject.transform.position +
                cam.transform.rotation * Vector3.forward, cam.transform.rotation *
Vector3.up);

            hpBarAmount.fillAmount = currHP / maxHP;

            if(currHP <= 0)
            {
                Die();
            }
        }

        private void Die()
        {
            if (isPlayerUnit)
            {

InputManager.InputHandler.instance.selectedUnits.Remove(gameObject.transform.parent);
                Destroy(gameObject.transform.parent.gameObject);
            }
            else
            {
```

```
                Destroy(gameObject.transform.parent.gameObject);
            }

        }
    }
}



using UnityEngine;
using UnityEngine.AI;

namespace DIMA.Units.Player
{
    [RequireComponent(typeof(NavMeshAgent))]
    public class PlayerUnit : MonoBehaviour
    {
        private NavMeshAgent navAgent;

        [HideInInspector]
        public UnitStatTypes.Base baseStats;

        public BasicUnit unitType;
        public UnitStatDisplay statDisplay;

        private void Start()
        {
            baseStats = unitType.baseStats;
            statDisplay.SetStatDisplayBasicUnit(baseStats, true);
            navAgent = GetComponent<NavMeshAgent>();
            navAgent.speed = baseStats.speed;
        }

        public void MoveUnit(Vector3 destination)
        {
            if(navAgent == null)
            {
                navAgent = GetComponent<NavMeshAgent>();
                navAgent.speed = baseStats.speed;
                navAgent.SetDestination(destination);
            }
            else
            {
                navAgent.SetDestination(destination);
            }
        }
    }

}



using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;
using DIMA.Units.Player;

namespace DIMA.InputManager
{
    public class InputHandler : MonoBehaviour
    {
        public static InputHandler instance;
```

```
        public List<Transform> selectedUnits = new List<Transform>();

        public Transform selectedBuilding = null;

        public LayerMask interactableLayer = new LayerMask();

        private bool isDragging = false;

        private RaycastHit hit;
        private Vector3 mousePosition;

        void Awake()
        {
            instance = this;
        }

        private void OnGUI()
        {
            if (isDragging)
            {
                Rect rect = MultiSelect.GetScreenRect(mousePosition, Input.mousePosition);
                MultiSelect.DrawScreenRect(rect, new Color(0f, 0f, 0f, 0.25f));
                MultiSelect.DrawScreenRectBorder(rect, 3,Color.blue);
            }
        }

        public void HandleUnitMovement()
        {
            if (Input.GetMouseButtonDown(0) &&
!EventSystem.current.IsPointerOverGameObject())
            {
                mousePosition = Input.mousePosition;

                Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
                if (Physics.Raycast(ray, out hit, 100, interactableLayer))
                {
                    if (addedUnit(hit.transform, Input.GetKey(KeyCode.LeftShift)))
                    {

                    }
                    else if (addedBuilding(hit.transform))
                    {

                    }
                }
                else
                {
                    isDragging = true;
                    DeselectUnits();
                }
            }
            if (Input.GetMouseButtonUp(0))
            {
                foreach (Transform child in Player.PlayerManager.instance.playerUnits)
                {
                    foreach (Transform unit in  child)
                    {
                        if (IsWhithinSelectionBounds(unit))
                        {
                            addedUnit(unit, true);
```

```
                        }

                    }
                }
                isDragging = false;
            }
            if (Input.GetMouseButtonDown(1) && HaveSelectedUnits())
            {
                mousePosition = Input.mousePosition;
                Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
                if (Physics.Raycast(ray, out hit))
                {
                    LayerMask layerHit = hit.transform.gameObject.layer;
                    switch (layerHit.value)
                    {
                        case 9:

                            break;
                        case 10:
                            //attack
                            break;
                        default:
                            foreach(Transform unit in selectedUnits)
                            {
                                PlayerUnit pUnit =
unit.gameObject.GetComponent<PlayerUnit>();
                                pUnit.MoveUnit(hit.point);
                            }
                            break;
                    }
                }
            }
            else if(Input.GetMouseButtonDown(1) && selectedBuilding != null)
            {

selectedBuilding.gameObject.GetComponent<Interactables.IBuildings>().SetSpawnMarkerLocation(
);
            }
        }

        private void DeselectUnits()
        {
            if (selectedBuilding)
            {

selectedBuilding.gameObject.GetComponent<Interactables.IBuildings>().OnInteractExit();
                selectedBuilding = null;
            }
            for (int i = 0; i < selectedUnits.Count; i++)
            {

selectedUnits[i].gameObject.GetComponent<Interactables.IUnits>().OnInteractExit();
            }
            selectedUnits.Clear();
        }

        private bool IsWhithinSelectionBounds(Transform obj)
        {
            if (!isDragging)
            {
                return false;
```

```
            }

            Camera cam = Camera.main;
            Bounds vpBounds = MultiSelect.GetVPBounds(cam, mousePosition,
Input.mousePosition);
            return vpBounds.Contains(cam.WorldToViewportPoint(obj.position));
        }

        private bool HaveSelectedUnits()
        {
            if (selectedUnits.Count > 0)
            {
                return true;
            }
            else return false;
        }

        private Interactables.IUnits addedUnit(Transform tf, bool canMultiselect = false)
        {
            Interactables.IUnits iUnit = tf.GetComponent<Interactables.IUnits>();
            if (iUnit)
            {
                if (!canMultiselect)
                {
                    DeselectUnits();
                }

                selectedUnits.Add(iUnit.gameObject.transform);

                iUnit.OnInteractEnter();

                return iUnit;
            }
            else
            {
                return null;
            }
        }

        private Interactables.IBuildings addedBuilding(Transform tf)
        {
            Interactables.IBuildings iBuilding =
tf.GetComponent<Interactables.IBuildings>();

            if (iBuilding)
            {
                DeselectUnits();
                selectedBuilding = iBuilding.gameObject.transform;
                iBuilding.OnInteractEnter();
                return iBuilding;
            }
            else
            {
                return null;
            }
        }
    }
}
```

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

namespace DIMA.UI.HUD
{
    public class ActionFrame : MonoBehaviour
    {
        public static ActionFrame instance = null;

        [SerializeField] private Button actionButton = null;
        [SerializeField] private Transform layoutGroup = null;

        private List<Button> buttons = new List<Button>();
        private PlayerActions actionsList = null;

        public List<float> spawnQueue = new List<float>();
        public List<GameObject> spawnOrder = new List<GameObject>();

        public GameObject spawnPoint = null;


        private void Awake()
        {
            instance = this;
        }

        public void SetActionButtons(PlayerActions actions, GameObject spawnLocation)
        {
            actionsList = actions;
            spawnPoint = spawnLocation;

            if (actions.basicUnits.Count > 0)
            {
                foreach (Units.BasicUnit unit in actions.basicUnits)
                {
                    Button btn = Instantiate(actionButton, layoutGroup);
                    btn.name = unit.name;
                    GameObject icon = Instantiate(unit.icon, btn.transform);
                    buttons.Add(btn);
                }
            }

            if (actions.basicBuildings.Count > 0)
            {
                foreach (Buildings.BasicBuilding building in actions.basicBuildings)
                {
                    Button btn = Instantiate(actionButton, layoutGroup);
                    btn.name = building.name;
                    GameObject icon = Instantiate(building.icon, btn.transform);
                    buttons.Add(btn);
                }
            }
        }

        public void ClearActions()
        {
            foreach (Button btn in buttons)
            {
                Destroy(btn.gameObject);
            }
```

```
    buttons.Clear();
}

public void StartSpawnTimer(string objectToSpawn)
{
    if (IsUnit(objectToSpawn))
    {
        Units.BasicUnit unit = IsUnit(objectToSpawn);
        spawnQueue.Add(unit.spawnTime);
        spawnOrder.Add(unit.playerUnitPrefab);
    }
    else if (IsBuilding(objectToSpawn))
    {
        Buildings.BasicBuilding building = IsBuilding(objectToSpawn);
        spawnQueue.Add(building.spawnTime);
        spawnOrder.Add(building.buildingPrefab);
    }
    else
    {
        Debug.Log($"{objectToSpawn} is not a spawnable object");
    }

    if (spawnQueue.Count == 1)
    {
        ActionTimer.instance.StartCoroutine(ActionTimer.instance.SpawnQueueTimer());
    }
    else if (spawnQueue.Count == 0)
    {
        ActionTimer.instance.StopAllCoroutines();
    }
}

private Units.BasicUnit IsUnit(string name)
{
    if (actionsList.basicUnits.Count > 0)
    {
        foreach (Units.BasicUnit unit in actionsList.basicUnits)
        {
            if (unit.name == name)
            {
                return unit;
            }
        }
    }
    return null;
}

private Buildings.BasicBuilding IsBuilding(string name)
{
    if (actionsList.basicBuildings.Count > 0)
    {
        foreach (Buildings.BasicBuilding building in actionsList.basicBuildings)
        {
            if (building.name == name)
            {
                return building;
            }
        }
    }
    return null;
}
```

```
        public void SpawnObject()
        {
            GameObject spawnedObject = Instantiate(spawnOrder[0], new
Vector3(spawnPoint.transform.parent.position.x,
                spawnPoint.transform.parent.position.y,
spawnPoint.transform.parent.position.z), Quaternion.identity);

            Units.Player.PlayerUnit pu =
spawnedObject.GetComponent<Units.Player.PlayerUnit>();
            pu.transform.SetParent(GameObject.Find("Player " + pu.unitType.type.ToString() +
"s").transform);


spawnedObject.GetComponent<Units.Player.PlayerUnit>().MoveUnit(spawnPoint.transform.position
);
            spawnQueue.Remove(spawnQueue[0]);
            spawnOrder.Remove(spawnOrder[0]);

        }
    }
}
```