МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

Кваліфікаційна робота бакалавра

на тему: **Розробка гри DINO**

Виконав студент групи **К-21і**
спеціальності 122 Комп'ютерні науки
Яковенко Максим Олегович

Керівник **ст. викладач**
Штефан Наталія Зінов'ївна

Консультант **док. техн. наук,**
професор
Казакова Надія Феліксівна

Рецензент **к.т.н., доцент**
Сергієнко Андрій
Володимирович

Одеса 2023

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА ТЕРМІНІВ

UI – User Interface

UE – Unreal Engine

Крафт – це механіка створення предметів

Спрайти – це зображення, які представляють активи гри

Пресет (preset) – це заздалегідь встановлені значення, шаблон або налаштування

C++ – мова програмування

C# – мова програмування

Java – мова програмування

Windows – операційна система

Unity – двигун для розробки ігрових додатків

ВСТУП

Комп'ютерні ігри сьогодні є популярними серед людей завдяки своїй розважальності, можливості втекти в інші віртуальні світи, відчуттю досягнень та прогресу, соціальній взаємодії з іншими гравцями, можливості випробувати нові ролі та жанри, а також відповіді на емоційні потреби, такі як захоплення, випробування навичок та вирішення викликів.

Якщо порівнювати мобільні ігри з іграми для ПК, геймери віддають перевагу ПК. Це через розмір екрану. До ПК можна підключити 42-дюймовий дисплей 4K. Однак це не те, що можна зробити на мобільному телефоні. Крім того, новітні графічні процесори підтримують трасування променів у реальному часі та ефекти тіні. Ці функції допомагають зробити ігровий процес реальним на ПК. Мобільні ігри пропонують хорошу графіку, але якщо ви не використовуєте гарнітуру віртуальної реальності, все це виглядає нереально.

Roguelike ігри – це жанр відеоігор, відомий своєю складною та непередбачуваною природою. Основна ідея полягає в тому, що коли персонаж гине в грі, гравцю доведеться починати все спочатку. Рівні, предмети та вороги генеруються випадковим чином щоразу, коли ви граєте, що робить кожну гру свіжою та унікальною. Це все про прийняття стратегічних рішень, управління обмеженими ресурсами та адаптацію до несподіваних ситуацій. Roguelikes відомі як жорсткі, але корисні, пропонуючи захоплюючі пригоди з кожним проходженням [1].

Мета дипломної роботи – розробка ігрового у стилі «Roguelike» з використанням 2D графіки у рушії Unity.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Опис предметної області

Існує кілька категорій гравців: хардкорні, мідкорні та казуальні. Основний упор розробників ігор завжди був спрямований на них.

Витоки ігор «Roguelike» можна простежити до «Rogue» – комп'ютерної гри , написаної 1980 року. Її основною темою є дослідження підземель. Вона була надзвичайно популярною на університетських Unix-системах на початку 1980-х років і породила цілий жанр, відомий сьогодні як Roguelike.

Сучасні Roguelikes набули популярності, оскільки вони привертають увагу ширшої аудиторії геймерів. Традиційні Roguelike ніколи не увійшли в мейнстрім частково тому, що гравцям не подобається відчуття повної втрати, починаючи нову гру. Крім того, коли гравці більше звикають до ігор, вони можуть оцінити основні аспекти roguelikes: рандомізовані рівні та permadeath.

Рандомізовані рівні зберігають кожне проходження свіжим і дозволяють гравцям вирішувати будь-які ігрові завдання, сприяючи творчому підходу до вирішення проблем. Permadeath надає відчуття значущості кожному проходженню та змушує гравців адаптувати свої стратегії до напруги втрати свого персонажа. Ці два елементи потужні самі по собі, але в поєднанні вони створюють дуже популярний жанр. Не кожна гра з рандомізованими рівнями та permadeath обов’язково хороша, але ці елементи значно покращують те, що робить багато відеоігор привабливими: виклик і відкриття [1].

Останнім часом створення roguelike стало популярним серед розробників. Жанр пережив відродження, і багатьох інді-розробників приваблюють виклики та креативність, які він пропонує. Процедурна генерація рівнів і механіка permadeath забезпечують унікальний ігровий процес і захоплюють гравців. Крім того, ігри типу roguelike часто заохочують до експериментів, оскільки гравці вчаться з кожного проходження, щоб покращити свої навички. Популярність roguelike можна побачити в численних

успішних назвах, випущених за останні роки, таких як Don't Starve, The Binding of Isaac, Darkest Dungeon і RimWorld, і розробники та гравці постійно цікавляться вивченням потенціалу жанру.

Roguelike відомі тим, що змушують гравців думати, стратегічно керувати ресурсами, вони візуально привабливі та викликають відчуття задоволення після успіху в квестах або викликах, особливо якщо вони були особливо складними. Вони чудово підходять для геймерів, які шукають додаткових завдань і хочуть перевірити свої здібності та навички [2].

1.2 Характеристика об’єкту розробки

Є декілька основних вимог при проектуванні ігор «Roguelike», які слід враховувати:

1. Процедурна генерація: Важливо мати механізм, який генерує випадкові рівні, предмети, ворогів та інші складові гри. Це забезпечує унікальність і різноманітність кожної гри, дозволяючи гравцям досліджувати нові середовища та стикатися з випадковими викликами.
2. Permadeath: При втраті персонажа гравці повинні починати гру спочатку. Це створює відчуття напруження та значимості кожного рішення, адже втрата персонажа означає втрату всього прогресу. Permadeath спонукає гравців бути обережними та ретельно планувати свої дії.
3. Стратегічний геймплей: Roguelike ігри наголошують на стратегічних рішеннях. Гравці повинні розумно використовувати свої ресурси, розраховувати ходи та адаптуватися до змінюючихся умов. Це може включати тактику бою, управління інвентарем, вибір шляху до мети та взаємодію з середовищем.
4. Високий рівень складності: Roguelike ігри відомі своєю високою складністю. Вони вимагають від гравців навичок, стратегії та вміння

приймати швидкі рішення. Це створює виклик та може принести велике задоволення від перемоги над складними ворогами або ситуаціями.

5. Прогресія та розблокування: Деякі Roguelike включають системи прогресії, які дозволяють гравцям отримувати нові навички, предмети або персонажів під час кожної гри. Це стимулює гравців до подальшого геймплею і дає відчуття постійного розвитку.
6. Стилізована графіка та атмосфера: Велика увага приділяється візуальному виконанню та створенню особливої атмосфери. Багато Roguelike мають піксельну графіку або відтворюють стилістику ретро-ігор, що надає їм унікальний вигляд та чарівність [3].

1.3 Огляд аналогів

При проектуванні інформаційної системи слід враховувати цілі, для яких створюється програмний продукт, умови, в яких експлуатуватиметься система, вимоги майбутніх користувачів

Для початку слід розглянути декілька топових аналогів, щоб з’ясувати головні від’ємності та збіжності ігор. Перший аналог це гра «The Binding of Isaac» – легендарна гра, що отримала широке визнання від гравців і критиків за свою високу переіграбельність, різноманітність та складність. "The Binding of Isaac" стала популярною серією, а її успіх призвів до випуску доповнень та перезавантаження гри з покращеною графікою та новими контентом – головоломка, яку люблять мільйони гравців у всьому світі (рис. 1).

У грі "The Binding of Isaac" головний акцент робиться на постійній смерті (permadeath) та процедурній генерації рівнів, що забезпечують унікальність кожного проходження. Гравцеві потрібно битися з ворогами, збирати різноманітні предмети, які надають йому різні здібності та уміння, і прогресувати крізь підвал, знаходячи босів і різні секрети. Гра відзначається своїм мрачним арт-стилем та тематикою, що включає багато релігійних і

символічних елементів. "The Binding of Isaac" пропонує глибокий ігровий процес з великою кількістю комбінацій предметів і варіативність вибору шляху гравця [4].

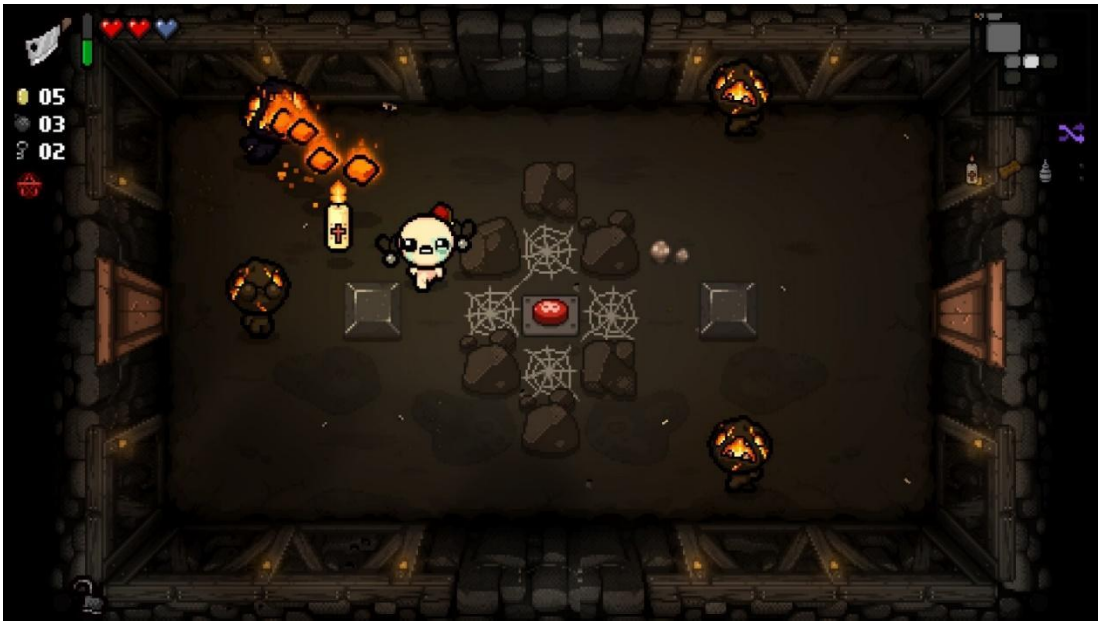

Рисунок 1 – Скріншот гри «"The Binding of Isaac»

До особливостей «"The Binding of Isaac» можна віднести:

– Різноманітність предметів: Гра має велику кількість різних предметів, які гравець може зібрати. Кожен предмет надає Айзеку нові здібності, міняючи геймплей та стратегію. Комбінування предметів може привести до потужних ефектів і унікального стилю гри.
– Випадкова генерація рівнів: Кожне проходження гри створює нові випадково згенеровані рівні. Це означає, що кожна гра буде унікальною з новими ворогами, кімнатами, пастками та секретами. Гравцю доведеться адаптуватися до нових умов і розраховувати свої дії на ходу.
– Багато різних босів: Гра має велику кількість різних босів, кожен з яких має свої унікальні атаки та поведінку. Битися з босами вимагає

стратегічного мислення та вміння швидко реагувати на їхні рухи.

– Тематика і атмосфера: "The Binding of Isaac" відзначається своїм мрачним, химерним стилем та тематикою, яка базується на біблійній історії про Абрахама та його сина Ісака. Гра створює особливу атмосферу, яка поєднує жахливість і гумор.
– Висока переіграбельність: Завдяки випадковій генерації рівнів, широкому спектру предметів та можливості вибору різних шляхів, "The Binding of Isaac" має високу переіграбельність. Гравці можуть проводити безліч годин, експериментуючи з різними стратегіями та відкриваючи нові секрети.

«Darkest Dungeon» – це відома гра, у якій гравець виконує роль власника покинутого родового маєтку, який зазнав прокляття (рис. 2). Його завдання полягає в тому, щоб відновити маєток, побудувати команду героїв і протистояти жахливим тваринам, хворобам та психологічним стресам.

Рисунок 2 – Скріншоти рівнів гри «Darkest Dungeon»

Особливості "Darkest Dungeon" включають химерний світ з темною атмосферою, стратегічні битви пошагового типу, залежність від ресурсів та здоров'я персонажів, а також систему стресу, яка впливає на психологічний стан героїв.

Гра пропонує глибоку систему управління персонажами, де гравцеві доведеться керувати їхнім здоров'ям, стресом, навичками та екіпіруванням. Також важливим елементом є управління ресурсами, такими як золото, продовольство та ліки.

"Darkest Dungeon" відома своїм надзвичайно високим рівнем складності, де випробовується стратегічне мислення та уміння приймати ризиковані рішення. Вона ставить гравця перед численними випробуваннями, включаючи небезпечні битви, випадкові події та психологічний тиск на героїв.

"Darkest Dungeon" є викликом для тих, хто шукає складну гру зі змістом і незвичайною атмосферою. Вона пропонує глибокий геймплей і вражаючий візуальний стиль, створюючи непередбачувану та захоплюючу пригоду у світі жахів.

У грі гравець приймає роль командира, який керує групою героїв, що відправляються на небезпечні випробування в підземеллях. Головним завданням є виживання та подолання жахливих ворогів.

"Darkest Dungeon" створює моторошну атмосферу завдяки своїй унікальній графіці, темному саундтреку та насиченому сюжету про боротьбу зі своїми внутрішніми демонами.

«RimWorld» це це науково-фантастична гра про виживання, де гравець керує групою колоністів, які висаджуються на віддаленій планеті (рис. 3). У грі гравець будує і керує свою колонію, забезпечуючи колоністів ресурсами, проживанням та безпекою. Головна мета полягає в тому, щоб вижити в небезпечному середовищі, забезпечуючи свою колонію їжею, енергією та захистом від ворожих фракцій, диких тварин і природних катастроф.

Рисунок 3 – Скріншоти гри «RimWorld»

Особливості "RimWorld" включають процедурну генерацію світу, розмаїття подій та історій колоністів, систему розвитку навичок та взаємодії зі збірною групою персонажів. Гра пропонує глибокий геймплей, де гравець повинен приймати стратегічні рішення щодо розподілу ресурсів, виробництва, медицини, досліджень та взаємодії з іншими фракціями. Кожен персонаж має свою унікальну історію, характеристики та навички, що впливають на їх роль у колонії.

"RimWorld" надає гравцеві велику свободу в прийнятті рішень і формуванні власної історії. Гра відома своєю несподіваністю і

непередбачуваністю, де події можуть швидко змінювати хід гри і ставити гравця перед складними виборами.

Гра відома своїм невпинним розвитком і непередбачуваністю подій. Гравцеві доведеться стикатися з різними випробуваннями, такими як натуральні катастрофи, напади інших фракцій та внутрішні конфлікти колоністів.

"RimWorld" є захоплюючою і складною грою, яка випробовує ваші навички управління, стратегічне мислення та здатність адаптуватися до змінних умов.

«Don`t Starve» це незвичайна інді-гра в жанрі виживання та пригоди, розроблена студією Klei Entertainment. Гра відбувається в небезпечному світі, наповненому монстрами, загадковими створіннями і безліччю загроз (рис. 4).

У грі керуєте одним з більше ніж десятка головиих героїв, які потрапляють у загадковий світ із суворими умовами. Головна мета – вижити якомога довше, забезпечуючи героя необхідними ресурсами, їжею, притулком і боротьбою зі стихійними небезпеками.

Рисунок 4 – Скріншоти гри «Don`t Starve»

Гра пропонує гравцю велику відкритість у виборі стратегій виживання. Тут доведеться збирати ресурси, мисливствувати, вирощувати їжу, будувати притулки та досліджувати небезпечний світ, щоб з'ясувати його таємниці. У грі присутній елемент рандомізації, що означає, що кожна гра буде унікальною, з новими випробуваннями та відкриттями.

"Don`t Starve" привертає увагу гравців своїм особливим мистецьким стилем, що нагадує малюнки на папері, та музикою, яка створює атмосферу загадки і напруження. Гра також пропонує різні режими гри, включаючи режим виживання, пригодницький режим та кооперативний режим.

Після аналізу топових аналогів при подальшому проектуванні об’єкту розробки слід враховувати декілька основних критеріїв:

- різноманіття графічних елементів ігрового поля та основних спрайтів;
- нескладні рівні;
- можливість грати оффлайн;
- звукові ефекти до подій у грі.

1.4 Вимоги до роботи

Щоб гра з була цікавою, гравці повинні відчути весь спектр емоцій. Гравець не повинен розслаблятися на жодну хвилину, адже якщо він дасть слабину – може статися будь-що, і тоді гра буде закінчена. Гравці повинні раціонально використовувати час, ланшафт, ресурси та можливості. Намагаються знаходити спільну мову з персонажами, використовувати ланшафт для більш помірної тактики битви, думати коли краще вступити в бій, а коли просто прослизнути повз ворогів.

Гравець повинен відчути себе героєм у складній ситуації та повністю взяти відповідальність за персонажа, намагаючись розібратися у складних ситуаціях та зрозуміти що відбувається навколо. Гравець повинен

перевершити себе, і всі труднощі на шляху, щоб прийти до довгоочікуваного фіналу, або померши в спробах.

Тож, до основних вимог роботи слід віднести наступні:

1. Розглянути аналоги даного жанру для виявлення основних принципів побудови гри.
2. Обрати програмні засоби розробки.
3. Описати ідею гри.
4. Вибір графічного контенту.
5. Моделювання поведінки героя у ігровому просторі.
6. Програмна реалізація дипломної роботи.

1.5 Вибір ігрового рушія

Вибір засобів розробки для створення ігор є важливим кроком для розробників. В роботі були розглянуті декілька рущіїв для розробки ігор.

1.5.1 Unreal Engine

Unreal Engine 5 (UE5) є найпотужнішим і найінтегрованішим движком, який поєднує високоякісну графіку та візуальний дизайн разом із аудіо, освітленням, анімацією та іншими можливостями.

Зокрема, UE5 дозволяє значно покращити графічну точність, позбавляючи розробників необхідності окремо визначати, як освітлюється об’єкт, замінюючи універсальний механізм освітлення, і рівень видимої деталізації динамічно зменшуючи модель найвищої точності. Саме вони складають велику частину роботи, щоб зробити гру гарною; Для створення динамічного освітлення потрібно багато роботи, і дизайнерам часто доводиться створювати кілька версій кожного об’єкта та персонажа з різними рівнями деталізації [4].

Є також вбудовані системи анімації та звуковий дизайн, які підключаються безпосередньо до інших частин механізму, тому вам не потрібно турбуватися про імпортування своєї роботи з іншого інструменту. Окрім іншого, передбачені спеціальні процеси для створення переконливих людських облич і тіл. (рис. 5).

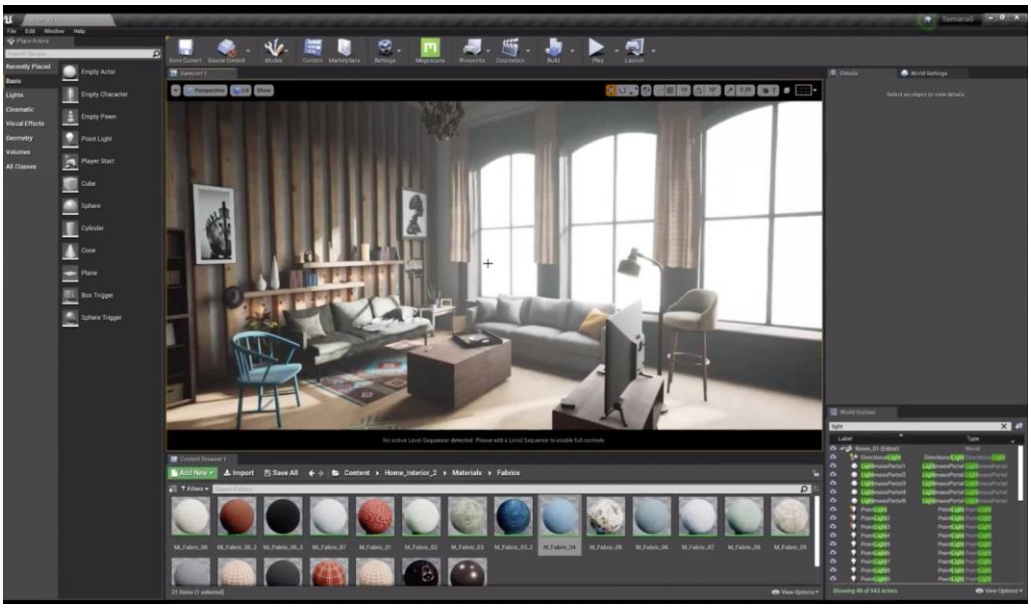

Рисунок 5 – Приклад етапу розробки гри у Unreal Engine

1.5.2 CryEngine

CryEngine є потужним ігровим рушієм, розробленим компанією Crytek. Він використовується для створення високоякісних ігор на різних платформах, таких як ПК, консолі та мобільні пристрої.

Основні особливості цього рушія полягають в його здатності до фотореалістичного рендерингу, високоякісного освітлення та деталізації оточуючого середовища.

CryEngine надає розробникам широкі можливості в галузі графічного дизайну, дозволяючи створювати деталізовані світи з реалістичними текстурами, ефектами частинок та шейдерами. Рушій підтримує фізичну

симуляцію, що дозволяє реалістично змоделювати рух об'єктів, руйнування та колізії.

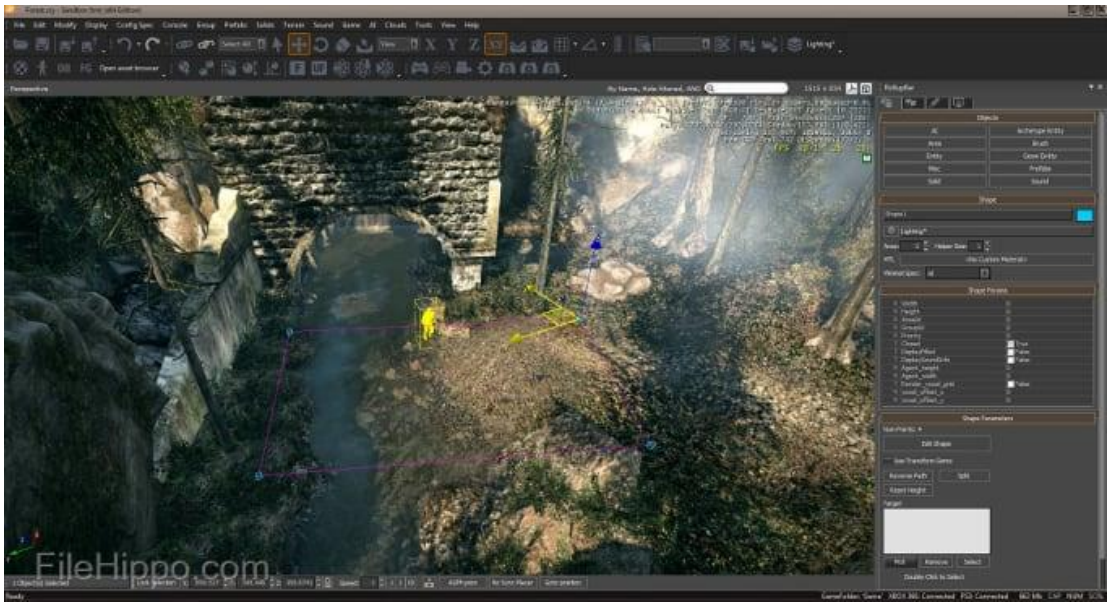

Рисунок 6 – Приклад етапу розробки гри у CryEngine

У CryEngine також присутня підтримка мультиплеєра, що дозволяє розробникам створювати онлайн-ігри зі збалансованою системою мультиплеєрних режимів. Рушій має потужний інструментарій розробки, який дозволяє розробникам зручно створювати і редагувати різноманітні об'єкти, управляти анімацією персонажів та створювати складні інтерактивні системи [5].

Одним з головних переваг CryEngine є його фотореалістичний рендеринг. Він використовує передові технології освітлення, такі як обчислення глобального освітлення (Global Illumination) і просвітлення подвійної точності (Double Precision Lighting), що дозволяють створювати деталізовані та реалістичні графічні ефекти [6].

Загальна гнучкість та розширюваність CryEngine дозволяють розробникам створювати ігри різних жанрів, від шутерів від першої особи до відкритих світів з великою свободою дій. Рушій також підтримує

використання мови програмування C++ для створення складних логічних систем та інтерфейсів.

Загалом, CryEngine є потужним ігровим рушієм з високим рівнем графічної якості, фізичною симуляцією та мультиплеєрною підтримкою. Він надає розробникам широкі можливості для створення захоплюючих ігрових світів, які вражають своєю реалістичністю та деталізацією.

1.5.3 Unity

Але не проаналізувавши багато варіантів рушіїв, я вибрав Unity. Добре відомий як один із висококласних ігрових движків. Unity містить AR, VR, 2D і 3D ігрові інструменти, які розробники можуть розгортати на різних платформах, таких як мобільні пристрої, ПК, ігрові консолі та Інтернет (рис. 7).

Рушій має кросплатформену підтримку, що дозволяє розробникам створювати ігри, які працюють на різних пристроях з одного кодової бази. Це забезпечує ефективність та зручність розробки [7].

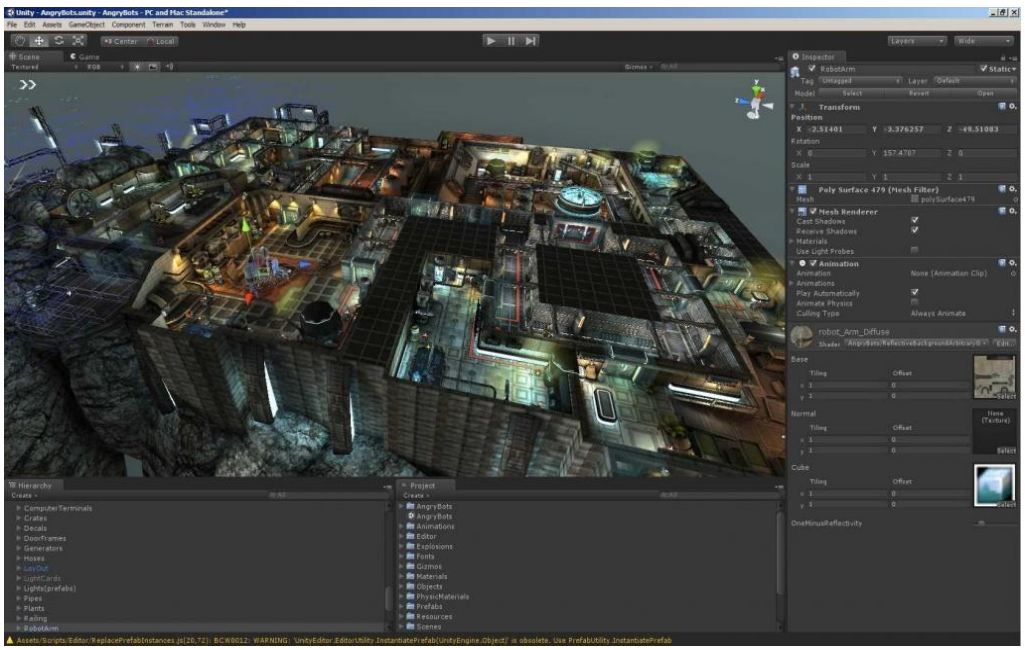

Рисунок 7 – Приклад етапу розробки гри у Unity

Основою Unity є мова програмування C#, яка використовується для створення функціональності та логіки ігрових об'єктів. Unity надає потужну інтегровану розробницьку середу, де розробники можуть зручно програмувати, налаштовувати параметри, редагувати візуальні ефекти та керувати ресурсами проекту.

Однією з ключових особливостей Unity є його графічний рушій. Він підтримує високоякісний 2D та 3D рендеринг, динамічні освітлення, тіні, ефекти частинок та фізичну симуляцію. Завдяки цьому, розробники можуть створювати деталізовані ігрові світи з реалістичною графікою та вражаючими візуальними ефектами.

Unity також має велику спільноту розробників, де можна знайти безліч ресурсів, плагінів та допомоги. Це сприяє активному обміну знаннями, вдосконаленню навичок та підтримці в процесі розробки [8].

1.6 Мова програмування

Мовою програмування була для проекту була вибрана C#, через підтримку в Unity, що з нами вже 22 роки. Мова програмування настільки універсальна та потужна, що легко протистоїть будь-якій іншій. Мова настільки повна, що сьогодні вона хоче увійти в історію: наразі це головний кандидат на звання мови програмування року.

Наразі C# стала однією з найуспішніших і широко використовуваних мов Microsoft, яка широко поширена серед професійних розробників. Хоча вона не є найпопулярнішою наразі вона входить до числа найпопулярніших мов програмування, поступаючись Python і C++. Однією з переваг C# є те, що його можна використовувати для створення програм як на мобільних, так і на комп’ютерних пристроях [9].

Спільнота з відкритим вихідним кодом також часто використовує його для створення програмного забезпечення за допомогою різних видів фреймворків, таких як Mono та .NET Framework [10].

C# все частіше використовується в корпоративному світі, і все, що Microsoft робить з ним, починає окупатися. Давайте подивимося, що і чому C# має хороші шанси стати найпопулярнішою мовою програмування 2023 року.

1.7 Сюжет гри

Зазвичай у ігр цієї категорії або майже повністю немає сюжету, або він надглибокий чи величезний. Саме в цьому він припадає на останній варіант. Головний герой – простий міщанин, що забіг у “Мурашник” – будівлю, яка живе по своїм власним законам часу та простору, і тепер йому треба боротись за своє життя.

Мурашник – це офіційно списана будівля – гуртожиток колегії чаклунів міста. За довгі роки будівля стала занадто небезпечною, щоб продовжити функціонування, тому її офіційно закрили і побудували нову. Але Мурашник не згинув у небуття. Через таємну обставину будівля поєднала у сові відображення всіх дивин, дефекти та аномалії, над якими працювали чаклуни.

Будівля змогла частково змінити простір всередині себе, раптово міняючи кімнати містами, розтягуючи, стискаючи їх, або навіть створювати нові, але які не завжди виходять нормальними. В деяких книги, столи, ліжка та їжа можуть бути зроблені зі скла або хітину.

Іноді будівля «викрадає» кімнати з різних будівель міста. Кімната може в один момент стиснутись до розмірів коробки з-під взуття і розчавити все, що було всередині в цей момент, про те навіть так, там все ще живуть люди. Можливо нескінченні перетинання коридорів – це чудове місце для несподіваних мешканців.

Всередині люди виживають як можуть: в одній за найбільш стабільних кімнат вони формували власне місто, де і живуть. Деякі з них – це потомки людей, що давно загубились тут і не змогли знайти вихід, залишившись тут і створивши тут сім'ю.

Головний герой попадає все в ослаблений мурашник, тому що демони назовні викачали з нього частину сили, герой повинен зрозуміти що сталося, і де він, і що взагалі треба робити. Він знаходить щоденники інших людей, що попали в цю пастку мурашника. Так герой дізнається, що десь є кімната з дверима, що можуть вивести його назовні, але через особливості будівлі, її знайти буде дуже складно.

Герой рано чи пізно може натрапити на саме місто, чи на людей, що направлять його туди. Місто назвали Їдальня, то воно розташоване у гіганській їдальні, що тягнеться на кілометри. В ній герой може знайти напів божевільного дослідника, допомогаючи якому можна пройти гру на одну з хороших кінцівок.

Через механіку перманентної смерті за відсутності можливості зберегти прогрес, гравець буде зацікавлений перепройти гру ще раз, але на іншу з кінцівок.

2 ПРОЕКТНА ЧАСТИНА

2.1 Генерація карти ігрового світу

Процедурно генеровані карти – базова особливість roguelike. Для жанру, який майже є синонімом поняття «випадковість» (і на те є причини), рандомізовані карти стали найпростішим способом демонстрації його ключового елемента, тому що вони впливають на багато аспектів геймплей – від стратегії дослідження та тактичного позиціонування до розташування предметів та ворогів.

В іграх Rogue-like мапа зазвичай може складатись з розгалуженого шляху у вигляді коридорів або кімнат та перешкод у звичайному вигляді, наприклад стіни, барикади, закриті двері, так і у вигляді сильних ворогів, деякі з котрих можна подолати щоб пройти далі, інші треба оминати .На шляху крім вже названих перешкод та ворогів можуть бути знайдені дружні сутності, наприклад торговці, та союзники, або речі, матеріали, зброя, монети , що допоможуть у подальшому розвитку та пригоді.

В Roguelike іграх з випадковою генерацією одним з найскладніших аспектів є сама процедура випадкової генерації карт. Зазвичай, карта складається з пресетів – наборів кімнат або невеликих модулів, які при старті гри випадковим чином з'єднуються між собою, утворюючи заплутані лабіринти, що кожного разу створюються унікально [7].

Процес генерації карт вимагає вирішення кількох важливих проблем, таких як забезпечення опанування маршрутів, балансування складності, створення різноманітних теренів та розміщення ресурсів. Розробники повинні забезпечити, щоб кожна згенерована карта була викликом для гравця і містила цікаві взаємодії, ворогів, секрети та можливості для дослідження.

Методи генерації можуть включати в себе використання алгоритмів, які засновані на випадкових числах, розміщенні об'єктів з урахуванням логіки та правил гри, а також використання параметрів, які визначають структуру та

вигляд карт. Важливо досягти балансу між випадковістю та геймплеєм, щоб карти були цікавими, але при цьому не ставали занадто непрохідними або незрозумілими для гравця.

Процедурна генерація карт в 2D іграх додає багато варіативності, переіграбельності та викликів для гравців, оскільки кожна нова гра пропонує унікальний світ для дослідження. Це сприяє стимулюванню гравців повертатись до гри знову і знову, даруючи їм нові пригоди та враження кожного разу, коли вони вирушають на пошуки нової генерованої карті.

На першому етапі проектування було створено декілька пресетів кімнат, а також було написано кілька скриптів на мові програмування C#. Карта була створена за допомогою процедурної генерації.

Кожного разу, коли гравець починає нову гру, карта генерується заново і не має схожості з попередніми версіями. Це забезпечує унікальність кожного проходження гри та надає гравцеві нові виклики і враження. Кожна нова карта може містити різноманітність теренів, об'єктів, ворогів та розміщення ресурсів, що робить гру більш цікавою і змушує гравця досліджувати нові області та приймати нові рішення.

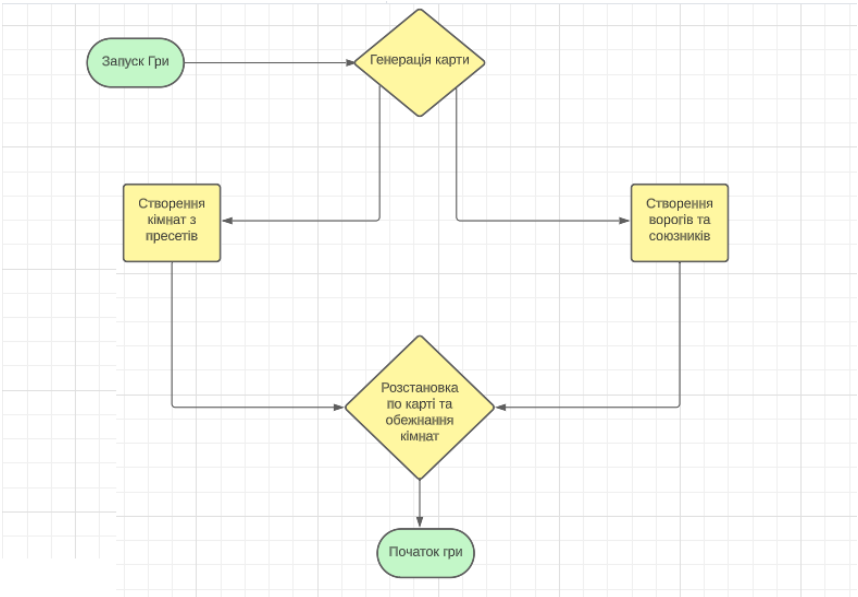

Рисунок 8 – Алгоритм генерації карти

Як показано на рисунку 8: при запуску гри, перш ніж гравець отримує управління персонажем, сама карта повинна згенеруватись, поки гравець чекатиме. В цей час активується алгоритм, що будує саму карту. Сама карта створена з багатьох об'єктів: ворогів, підлог, стін, бочок, та іншого. Об’єднати усе в групи, такими тут е кімнати. У одній може бути спальня кімната з ліжками, у іншій – пруд з монстрами.

Пресет (preset) – це заздалегідь встановлені значення, шаблон або налаштування, які можуть бути використані для швидкого налаштування або використання в програмному забезпеченні. Він може містити попередньо визначені параметри, настройки, конфігурації або варіанти, які дозволяють користувачам застосовувати готові налаштування замість налаштування кожного параметра окремо. Пресети забезпечують швидкий і зручний спосіб використання популярних налаштувань або стандартних конфігурацій для певної програми чи інструменту.

У Rogue-like іграх, з пресетів можуть наприклад створюватись кімнати або вороги. Для створення процедурно-генеруючої карти попередньо робиться пресет кімнати і зберігається. Наприклад ми зробимо кімнату з дверима зліва і знизу (LD) [8].

Цей пресет можна :

– підключити до генератора кімнат, і генератор зможе створювати необхідну їх кількість, утворивши з них карти;
– дублювати, створити ще один схожий пресет, який можна буде трохи переробити, а не робити новий з нуля;
– переробити : наприклад протягнувши дір зліва направо, і створити кімнату (RD);
– об'єднати з іншим пресетом;
– легко дороблять.

Можна навіть робити складові частини кімнат персетами, наприклад стіни, підлогу та мебель, що додасть ще більшого різноманіття створюваних кімнат.

Метою пресетів є спрощення роботи програмістів та гейм-дизайнерів, економія часу, та набагато більше варіантів справді випадково згенерованої рівней. Це один з основних інструментів для створення схожих проектів.

Приклад з’єднання пресетів показано на рисунку 9:

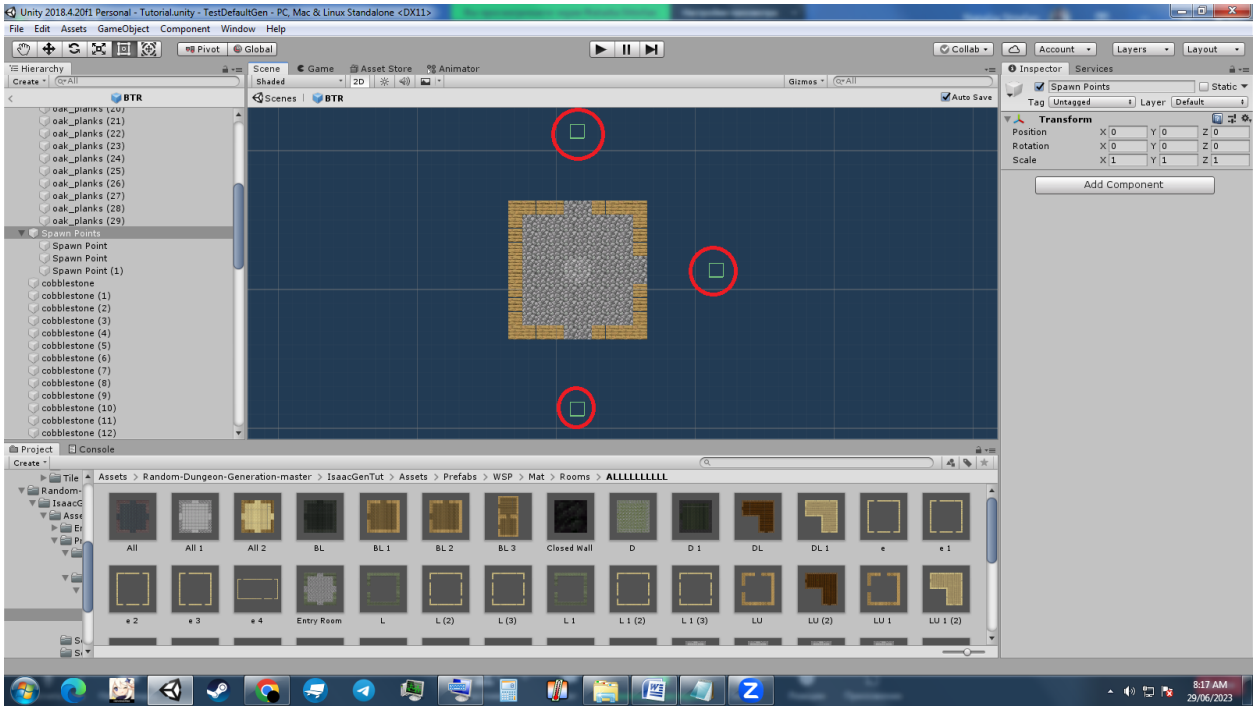

Рисунок 9 – Приклад з’єднання пресетів у Unity

У кожного пресету кімнати є мітки або маячки, які невидимі для гравця. Вони знаходяться біля дверних пройомів і відповідають за те, щоб кімнати з’єднувались між собою (рис. 10). У нього е дві сфери застосування:

- якщо наша кімната вже збудована, то маячки працюють у режимі створення нових кімнат на своєму місці;
- на а при виборі яка саме кімната буде збудована, вони перевіряють всі прсесети своєї категорії на наявність потрібного маячка, і тоді випадково вибирає один з підходящих .

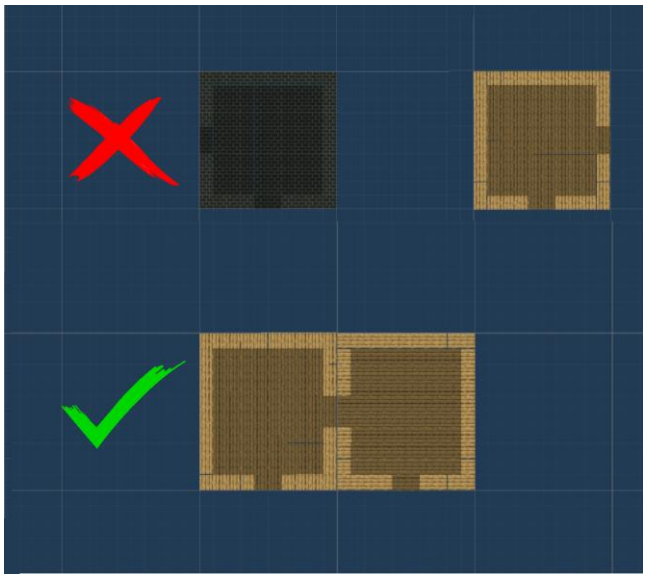

Рисунок 10 – Приклад з’єднання кімнат через тунель

Усе це потрібно для нормальної роботи генератора, щоб між кімнатами завжди був прохід, і вони могли правильно з’єднуватись між собою (рис. 11).

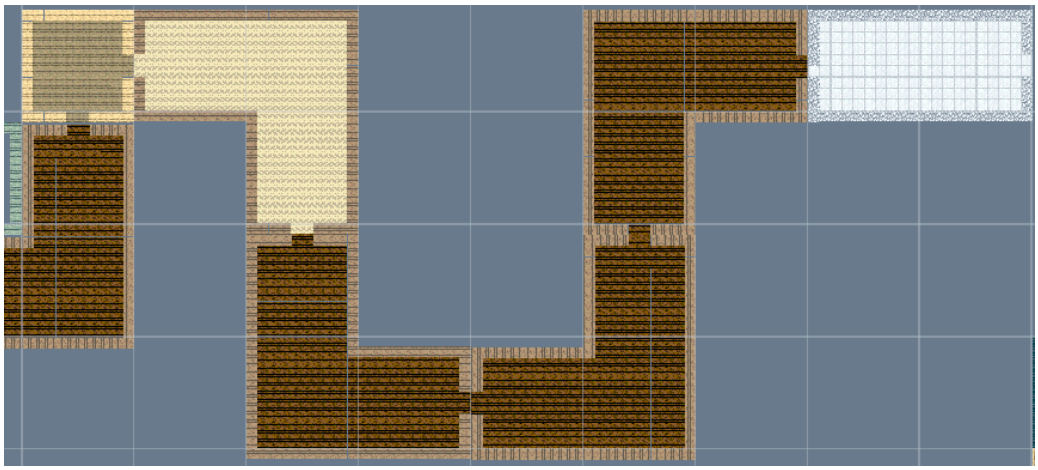

Рисунок 11 – Приклад вдалого з’єднання кімнат

Генерація карти автоматично переривається через визначену кількість секунд після початку, тому що без цього карта просто генерувалась би нескінченно, і менше ніж за хвилину це призвело б до критичної загрузки на процесору.

2.2 Взаємодія зі світом

Гравець повинен не лише йти до цілі, а й оглянутись навколо, досліджуючи своє оточення. Тут важливим пунктом є імерсивність – це стан, коли гравець повністю поглинутий у віртуальний світ гри і відчуває себе часткою цього світу. Це створюється завдяки реалістичному візуальному оформленню, звуковому супроводу, геймплею та взаємодії з оточенням. Імерсивність допомагає створити враження присутності гравця у грі, викликає емоції та забезпечує більш насичений та захоплюючий досвід гри. Це одна з ключових характеристик успішних ігор, оскільки допомагає залучити гравця і зберегти його у світі гри на тривалий час.

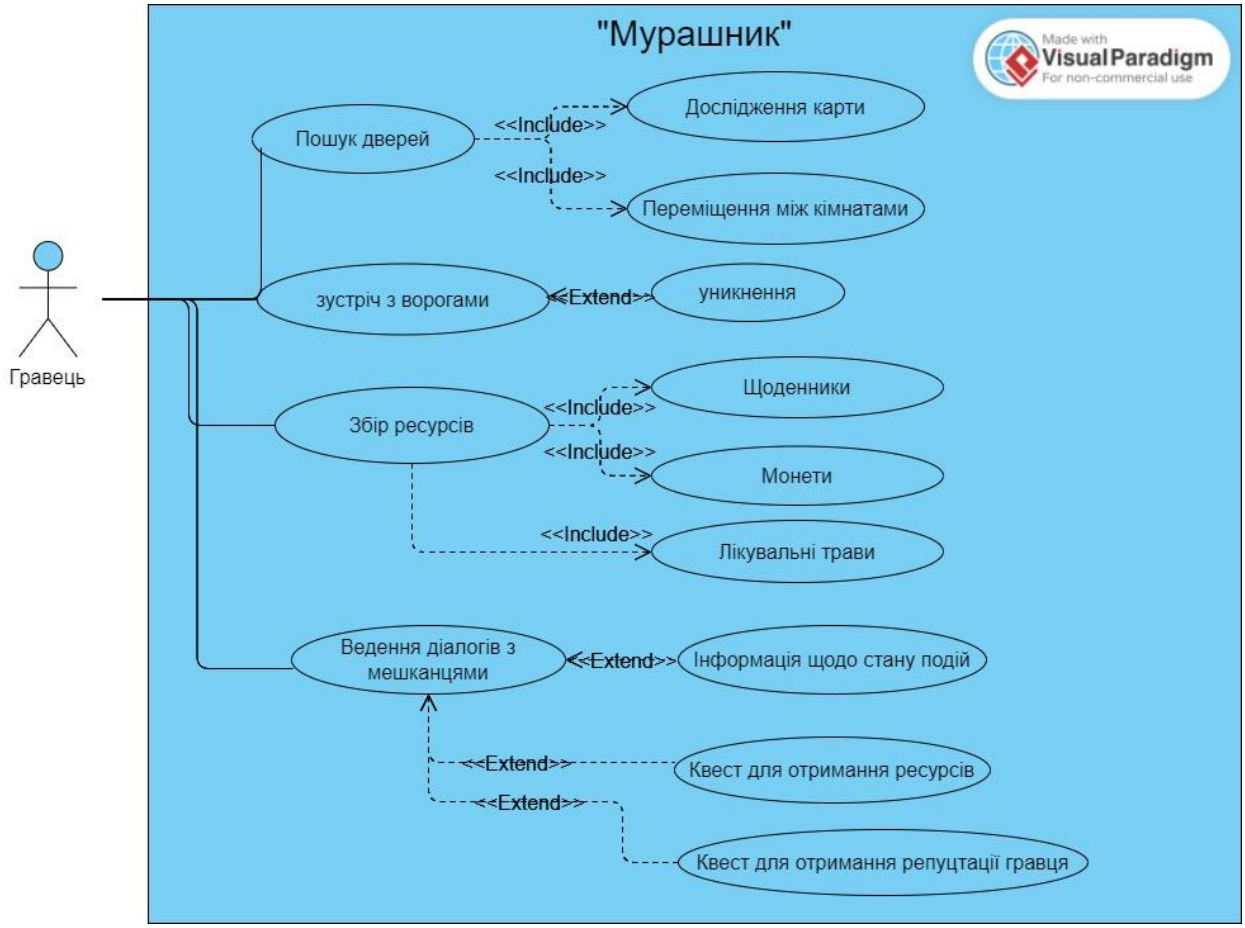

Рисунок 12 – Діаграма Use-Case для відображення взаємодії гравця з оточенням

На рисунку 12 зображено діаграму варіантів використання. Взаємодія з оточенням – це механіка, що значно оживляє будь-який проект. Наприклад гравець може розбивати стоячи бочки, сподіваючись знайти усередині якийсь ресурси ,що можуть бути йому у нагоді. Гравець може прочитати загублений щоденник, або залишену записку, щоб дізнатись щось про можливу небезпеку поруч, чи шматочок сюжету гри.

Пошук у загублених рюкзаках може привнести гравцю виграш в вигляді монет чи ресурсів, що можуть йому знадобитись, він може не знати нічого корисного, або ж знайти монстра або сутність, що ховалась у середині сумки, і лишитися або тікати, або прийняти битву.

Також гравець може взаємодіяти з союзними або нейтральними не ігровими героями, які надають додаткові можливості і завдання. Вони можуть надати гравцеві квести, наприклад, пошук певної кількості лікарських трав. За успішне виконання такого завдання, герой може отримати нагороду, наприклад, частину лікарської настоянки, яку вони приготували, а також отримати трохи монет. Крім того, гравець може здійснювати покупки та продаж своїх знайдених предметів від таких персонажів.

Вони можуть мати в своєму асортименті різноманітні предмети, які гравець може придбати за виграшну ціну або продати непотрібні артефакти для отримання додаткових ресурсів. Такий вид взаємодії додає глибину геймплею та можливості для розвитку персонажа. Гравець може планувати свої дії, вибирати, які завдання прийняти і які ресурси збирати для отримання більш значущих винагород. Це стимулює активне дослідження гри, взаємодію з різними персонажами та приносить відчуття розвитку і прогресу героя.

Через взаємодію з персонажами гри, гравець має можливість отримати додаткову інформацію про сюжет та глибше зануритися в ігровий світ. Кожен персонаж може мати свою унікальну історію, мотивацію та цілі, які можна дізнатись шляхом розмов з ними. Це дозволяє розкрити деталі сюжету, розвивати характери персонажів і створювати багатогранну інтригуючу історію.

Крім того, взаємодія з персонажами може впливати на подальший розвиток гри. Гравець може приймати рішення, які впливають на стосунки з персонажами, розкривають нові сюжетні лінії або відкривають можливість пройти гру на різних кінцівках. Це стимулює гравця до активного дослідження світу гри, виявлення секретів і взаємодії з різними персонажами, щоб дізнатись більше про унікальну історію, яка розгортається перед ним.

2.3 Проектування інтерфейсу

Взаємодія з оточенням у 2D іграх також включає інтуїтивний і мінімалістичний інтерфейс. Інтерфейс гри розроблений з метою забезпечити зручність гравця і не завантажувати його зайвою інформацією. Він добре продуманий, щоб не перекривати важливі частини гри і не відволікати гравця від головної дії. Інтерфейс є невидимим, коли це не потрібно, але доступним і зрозумілим, коли гравцю потрібно з ним взаємодіяти (рис. 13). Це дозволяє гравцеві максимально зосередитися на грі і погрузнути у її світ без зайвих перешкод.

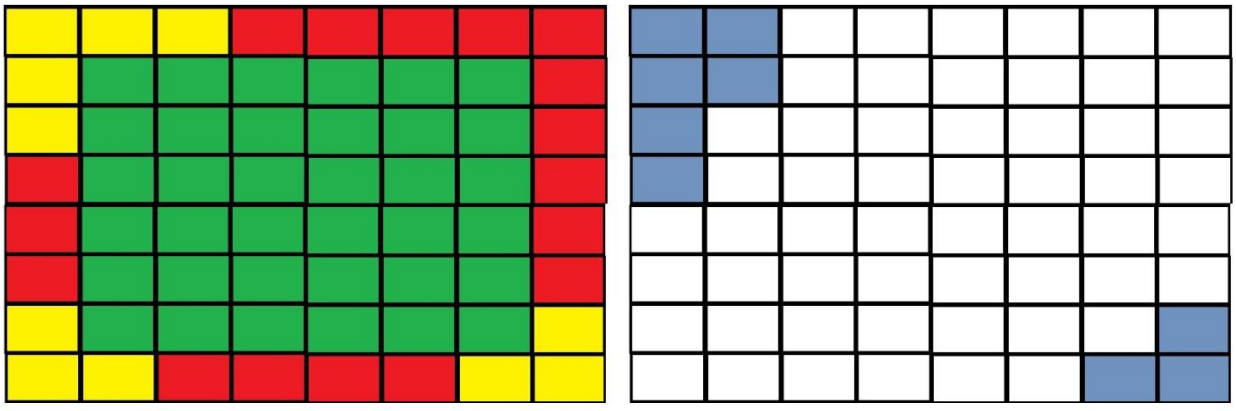

Рисунок 13 – Зони фокусування гравця в період гри

Перелік монет та інших ресурсів не повинен завжди бути перед очима, це не настільки важлива інформація, на яку треба постійно дивитися, але й ховати її не треба. Її зазвичай тримають поблизу центру, щоб на неї легко можна було переключити увагу та так же легко відвести її, переключившись на щось інше.

У дипломній роботі ця схема була сформована, опираючись на логіку, власний досвід, комфортність для гравця, та аналоги.

3 ОПИС РЕАЛІЗАЦІЇ ПРОЕКТУ

Ідея подібних ігор полягає в створенні динамічного та високозатратного ігрового досвіду для гравців. Головною метою таких ігор є створення випадкових та неповторних ігрових світів, які кожен раз, коли гравець розпочинає гру, будуть генеруватися новими способами.

Розробка такої гри вимагає створення широкого спектра компонентів, включаючи генерацію рівнів, систему управління персонажем, систему битви та взаємодії з оточенням. Додаткові елементи, такі як система розблокування рівнів або кімнат, прогресія персонажа, різноманітні вороги та предмети, додають глибину та варіативність до геймплею.

Для досягнення високої якості гри необхідно враховувати баланс геймплею, розробляти різноманітні завдання та виклики, а також створювати насичену атмосферу через візуальні ефекти, звуковий дизайн та музику.

Робота включає перелік графічних ресурсів та файли програмного коду. Структура проджект у Unity представлено на рисунку 14:

Рисунок 14 – Структура проекту Unity

3.1 Спрайти проекту

Для моделювання ігрового простору використовувались спрайти для кімнат та переходів, всі вони при генерації допомагають створити унікальні приміщення (рис. 15):

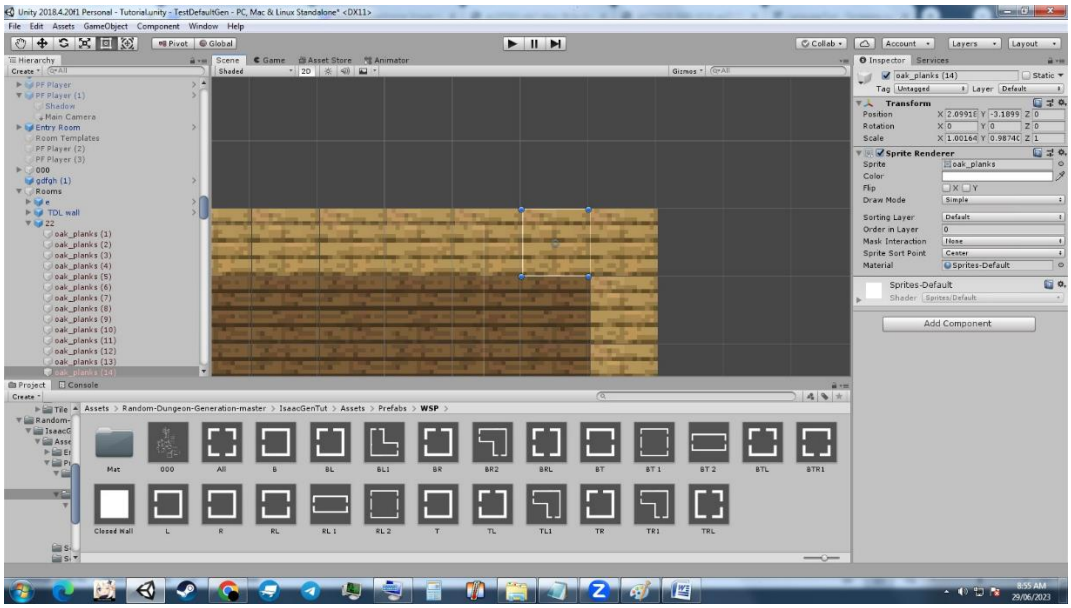

Рисунок 15 – Спрайти для елементів приміщень

Крім цього, у ігровому просторі присутні мешканці мурашника: чарівник та його помічник (рис. 16) та ворожі істоти (рис. 17).

Рисунок 16 – Спрайти чарівника та помічника

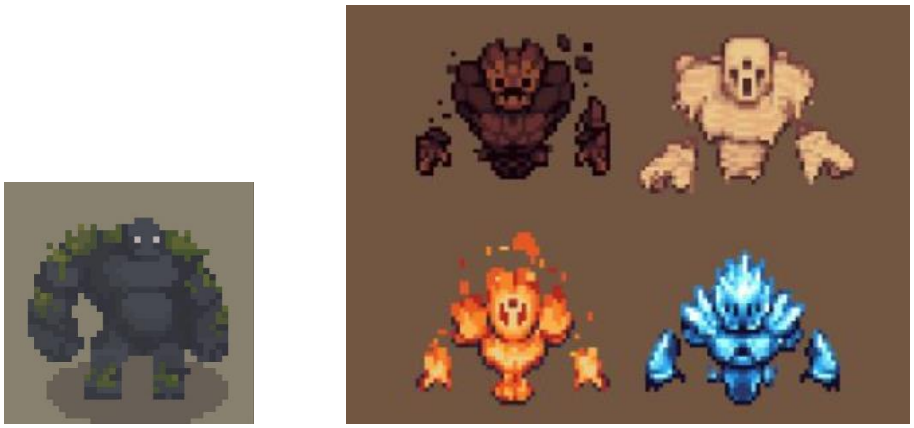

Рисунок 17 – Спрайти ворожих істот

Чарівник буде виконувати роль помічника, у якого гравець може взяти квест для отримання одного з успішного проходження гри. Після виконаного завдання гравцю буде доступна фінальна фаза гри.

Ворожі істоти є головними перешкодами на шляху до мети гравця. Кожен з них має свій вплив на персонажа, такий як, викрадання речей, заморозка або перекриття дверей до наступного приміщення.

3.2 Інтерфейс користувача

При запуску гри відкривається стартове меню (рис. 18). Після натискання на «Старт» генерується карта ігрового простору за завантажується для гравця. Є можливість ставити на паузу та повертатися до моменту гри і продовжувати її проходження.

Після старту гри, гравець опиняється у першому приміщенні (рис. 19). Тут є мешканці та сундук, який можна відкрити для отримання першого бонуса гри – монети. Але слід уникати зіткнення з летучими мишами, які зменшують рівень життєвих сил гравця.

Рисунок 18 – Стартове меню

Рисунок 19 – Скрін першої кімнати мурашника

При старті гри генерується карта ігрового простору, який гравець може бачити у себе на екрані (рис. 19).

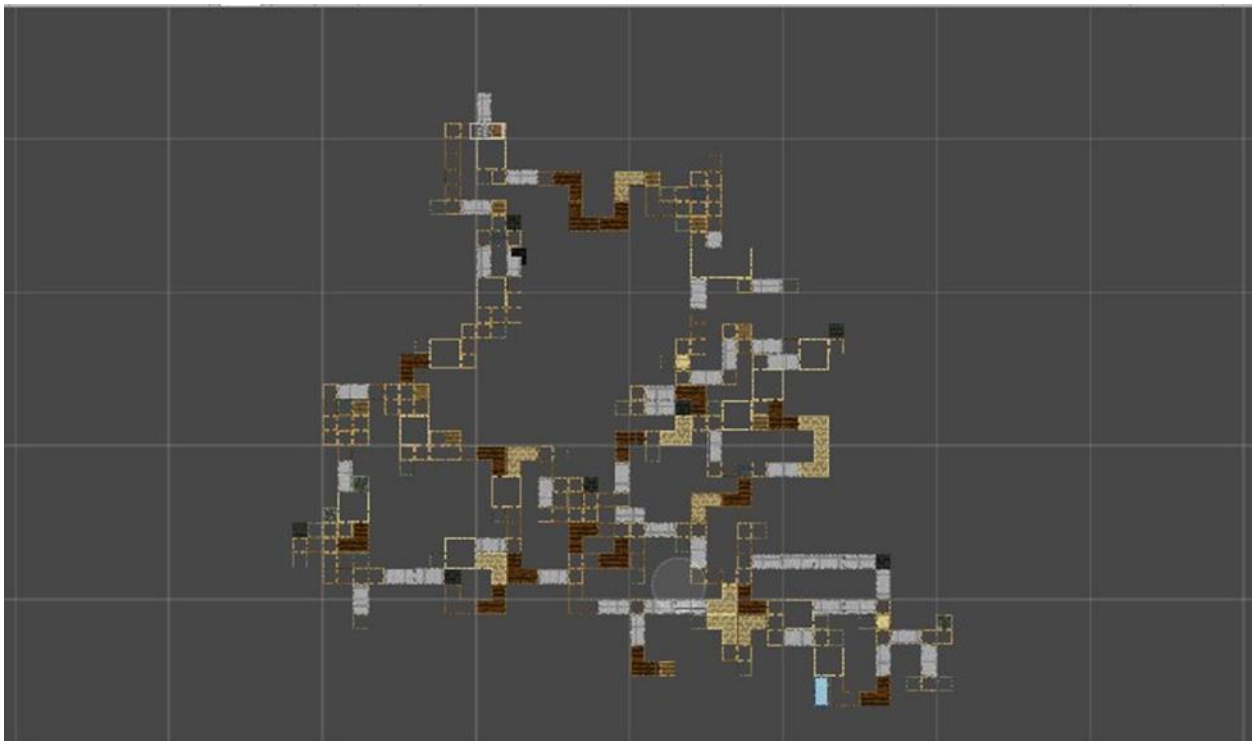

Рисунок 20 – Карта згенерованого левела

На рисунку 21 представлено скрін кімнати, де герой зустрічається з союзником, це чарівник.

Рисунок 21 – Кімната з чарівником

При зустрічі для гравця відображається вікно з діалогом, під час якого чарівник дає завдання на проходження квесту (рис. 22), це дозволить отримати додаткові бонуси:

Рисунок 22 – Скрін діалогу з чаклуном

На рисунку 23 представлено скрін сцени гри. У верхньому лівому куті розміщено меню гравця, а саме health-bar, тобто покажчик життєвих ресурсів. За замовчуванням, гравцю дається 4 життя. При зіткненні з ворогами, деякі з них забирають одне життя, але є і ті, хто відкачує половину життєвого ресурсу.

Крім того, послідовно проходячи по карті ігрового простору гравець має можливість збирати золоті монети, вони допоможуть відновити життєві ресурси (це колби з рідиною) або придбати додаткові артефакти для проходження квестів.

З урахуванням того, що гра генерує карти рівнів, не має сенсу ставити лічильник часу проходження рівня. За бажанням гравця, гру можна ставити на паузу і потім продовжувати проходження карти.

Рисунок 23 – Скрін сцени гри

3.3 Опис програмного коду

3.3.1 Головний клас «RoomSpawner»

Головний скрипт цього проекту: «RoomSpawner». В його структуру входять поля:

- для зберігання часу, з початку генерації карти;
- для перевірок на наявність міток або маячків;
- для активування міток;
- для перевірки можливості будувати нові кімнати;
- для деактивації маячків, коли вони нам вже стали непотрібні.

Спочатку слід підключити необхідні просторії імен та оголосити клас "RoomSpawner", що він клас "MonoBehaviour":

```
using System.Collections;
```

```
using System.Collections.Generic;
using UnityEngine;
public class RoomSpawner : MonoBehaviour
{
     public int openingDirection;
     // 1 --> потрібні двері внизу
     // 2 --> потрібні двері зверху
     // 3 --> потрібні двері зліва
     // 4 --> потрібні двері справа
     private RoomTemplates templates;
     private int rand;
     private bool spawned = false;

     public float waitTime = 4f;
     public float stopGenerationDelay = 2f;

private void Start()
{
```

Далі слід знайти об'єкт з тегом "Rooms" і отримати компоненту "RoomTemplates":

```
        templates =
GameObject.FindGameObjectWithTag("Rooms").GetComponent<RoomTempl
ates>();
```

Виклик методу "WaitAndStopGeneration" через певний час:

```
    StartCoroutine(WaitAndStopGeneration());
    // Виклик методу "Spawn" з невеликим затримкою
    Invoke("Spawn", 0.1f);
}
```

Корутин для затримки і вимкнення скрипта:

```
private IEnumerator WaitAndStopGeneration()
{
    yield return new WaitForSeconds(stopGenerationDelay);
    enabled = false;
}
```

Нам знадобиться метод для створення кімнати:

```
private void Spawn()
{
```

```
        if (spawned)
            return;
```

Отримання всіх коллайдерів, які перетинаються з даною позицією:

```
        Collider2D[] colliders =
Physics2D.OverlapCircleAll(transform.position, 1f,
LayerMask.GetMask("SpawnPoint"));
        foreach (Collider2D collider in colliders)
        {
            if (collider.gameObject != gameObject &&
collider.CompareTag("SpawnPoint"))
            {
```

Отримання іншого "RoomSpawner", якщо такий є:

```
                RoomSpawner otherSpawner =
collider.GetComponent<RoomSpawner>();

                if (otherSpawner != null &&
otherSpawner.spawned)
                {
```

Видалення поточного "RoomSpawner", якщо інший вже створений:

```
                    Destroy(gameObject);
                    return;
                }
            }
        }
```

Наступний крок, це створення кімнати залежно від напрямку виходу:

```
        if (openingDirection == 1)
        {
            rand = Random.Range(0,
templates.bottomRooms.Length);
            Instantiate(templates.bottomRooms[rand],
transform.position,
templates.bottomRooms[rand].transform.rotation);
        }
        else if (openingDirection == 2)
        {
            rand = Random.Range(0, templates.topRooms.Length);
```

```
            Instantiate(templates.topRooms[rand],
transform.position,
templates.topRooms[rand].transform.rotation);
        }
        else if (openingDirection == 3)
        {
            rand = Random.Range(0, templates.leftRooms.Length);
            Instantiate(templates.leftRooms[rand],
transform.position,
templates.leftRooms[rand].transform.rotation);
        }
        else if (openingDirection == 4)
        {
            rand = Random.Range(0, templates.rightRooms.Length);
            Instantiate(templates.rightRooms[rand],
transform.position,
templates.rightRooms[rand].transform.rotation);
        }

        spawned = true;
    }
```

Обробник події зіткнення з коллайдером:

```
    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("SpawnPoint"))
        {
```

Отримання іншого "RoomSpawner":

```
            RoomSpawner otherSpawner =
other.GetComponent<RoomSpawner>();

            if (otherSpawner != null && !otherSpawner.spawned &&
!spawned)
            {
```

Створення закритої кімнати і видалення поточного "RoomSpawner":

```
                Instantiate(templates.closedRoom,
transform.position, Quaternion.identity);
                Destroy(gameObject);
            }

            spawned = true;
        }}
```

3.2.2 Реалізація класу «RoomTemplates»

Цей клас потрібен для зберігання та взаємодії пресетів в ігровому рушії Unity. Тут є можливість обрати кількість комірок для подальшого генерування в них кімнат та їх типи для ігрового світу.

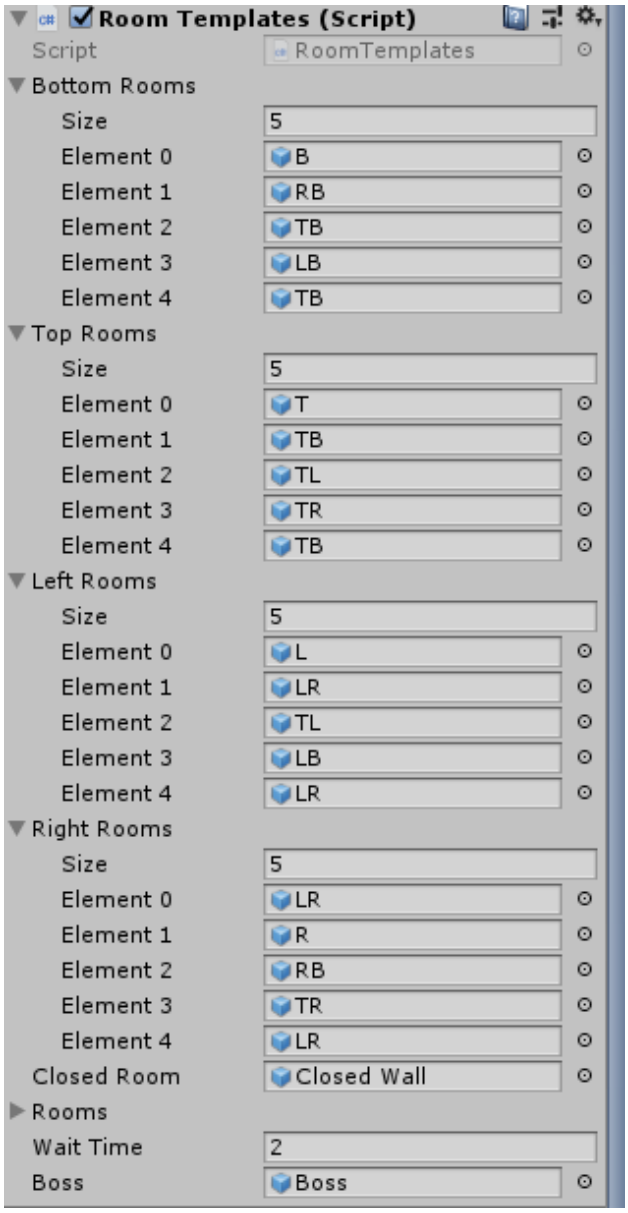

Рисунок 24 – RoomTemplates у панелі розробника Unity

Розглянемо реалізацію цього класу більш детально. По-перше, необхідно підключити необхідні простори імен:

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

Оголошення класу "RoomTemplates", який успадковує клас "MonoBehaviour".

Тут знадобляться такі об’єкти:

1. Масив об'єктів кімнат з дверима вниз.
2. Масив об'єктів кімнат з дверима зліва.
3. Масив об'єктів кімнат з дверима зліва.
4. Масив об'єктів кімнат з дверима справа.
5. Закрита кімната.
6. Список створених кімнат.
7. Час очікування перед створенням босса.
8. Прапорець, який позначає, чи вже створений босс.
9. Об'єкт босса.

```
public class RoomTemplates : MonoBehaviour {
public GameObject[] bottomRooms; // public GameObject[] topRooms; // Масив об'єктів кімнат з дверима зверху
public GameObject[] leftRooms; // Масив об'єктів кімнат з дверима зліва
public GameObject[] rightRooms; // Масив об'єктів кімнат з дверима справа
public GameObject closedRoom; // Закрита кімната
public List<GameObject> rooms; // Список створених кімнат
public float waitTime; // Час очікування перед створенням босса
private bool spawnedBoss; // Прапорець, який позначає, чи вже створений босс
public GameObject boss; // Об'єкт босса

void Update(){
if(waitTime <= 0 && spawnedBoss == false){
```

Прохід по всіх створених кімнатах:

```
            for (int i = 0; i < rooms.Count; i++) {
                if(i == rooms.Count-1){
```

Якщо поточна кімната - остання, створюємо босса в цій кімнаті:

```
                    Instantiate(boss,
rooms[i].transform.position, Quaternion.identity);
                    spawnedBoss = true;
                }
            }
        } else {
```

Зменшуємо час очікування:

```
            waitTime -= Time.deltaTime;
        }
        }
        }
```

3.2.2 Клас «AddRoom»

Підключення необхідних просторів імен та Оголошення класу "AddRoom", який успадковує клас "MonoBehaviour":

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

        public class AddRoom : MonoBehaviour {
        private RoomTemplates templates; // Змінна для збереження
посилання на об'єкт RoomTemplates

        void Start(){
```

Знаходимо об'єкт RoomTemplates у сцені, що має тег "Rooms" і отримуємо його компонент RoomTemplates:

```
        templates =
GameObject.FindGameObjectWithTag("Rooms").GetComponent<RoomTempl
ates>();
```

Додаємо поточний об'єкт кімнати в список кімнат*:*

```
RoomTemplates
templates.rooms.Add(this.gameObject);
}
}
```

3.3.3 Скрипт ворога

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

Ворог буде йти за гравцем. Спочатку слід задати швидкість переміщення ворога та межа знаходження відповідно до героя:

```
public class EnemyChase : MonoBehaviour
{
    public float moveSpeed = 3f;
    public float stoppingDistance = 1f;

    private Transform player;
```

Метод старт створює об’єкт героя, а метод оновлення перевіряє наближення ворога до героя.

```
        void Start()
        {
            GameObject playerObject =
GameObject.FindGameObjectWithTag("Player");
            if (playerObject != null)
            {
                player = playerObject.transform;
            }
            else
            {
                Debug.LogError("Player object not found!");
            }
        }
        void Update()
        {
            if (player == null)
```

```
            {
                return;
            }

            float distanceToPlayer =
Vector2.Distance(transform.position, player.position);

            if (distanceToPlayer > stoppingDistance)
            {
                MoveTowardsPlayer();
            }
        }

        void MoveTowardsPlayer()
        {
            transform.position =
Vector2.MoveTowards(transform.position, player.position,
moveSpeed * Time.deltaTime);
        }
    }
```

3.3.4 Скрипт пасивного мешканця

Пасивні мешканці не перетинаються шляхами з гравцем, а є більш як доповнення до візуалу сцен гри.

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

Тут слід ураховувати швидкість руху ворога, інтервал зміни напрямку руху, діапазон руху ворога, таймер для визначення інтервалу зміни напрямку, початкова позиція ворога та цільова позиція ворога:

```
    public class EnemyMovement : MonoBehaviour
    {
        public float moveSpeed = 3f; // Швидкість руху ворога
        public float changeDirectionInterval = 2f; // Інтервал
зміни напрямку руху
        public float movementRange = 3f; // Діапазон руху ворога

        private float timer; // Таймер для визначення інтервалу
зміни напрямку
```

```
        private Vector3 initialPosition; // Початкова позиція
ворога
        private Vector3 targetPosition; // Цільова позиція ворога
        private Animator animator; // Аніматор для зміни анімації
руху

        private static readonly int AnimParamX =
Animator.StringToHash("MoveX"); // Хеш-код параметра анімації
для горизонтального руху
        private static readonly int AnimParamY =
Animator.StringToHash("MoveY"); // Хеш-код параметра анімації
для вертикального руху
```

Для методу старту слід отримати нову цільову позицію, отримати компонент аніматора:

```
        void Start()
        {
            initialPosition = transform.position;
            GetNewTargetPosition(); // Отримати нову цільову
позицію
            animator = GetComponent<Animator>(); // Отримати
компонент аніматора
        }
```

Метод оновлення буде враховувати інтервал зміни напрямку та таймер:

```
        void Update()
        {
            timer += Time.deltaTime;

            if (timer >= changeDirectionInterval) // Якщо
інтервал зміни напрямку пройшов
            {
                GetNewTargetPosition(); // Отримати нову цільову
позицію
                timer = 0f; // Скинути таймер
            }

            MoveTowardsTarget(); // Рухатись до цільової позиції
            UpdateAnimation(); // Оновити анімацію руху
        }

        void GetNewTargetPosition()
        {
```

```
            float randomX = Random.Range(-movementRange,
movementRange); // Випадкова координата X в діапазоні руху
            float randomY = Random.Range(-movementRange,
movementRange); // Випадкова координата Y в діапазоні руху
            targetPosition = initialPosition + new
Vector3(randomX, randomY, 0f); // Оновити цільову позицію
        }
```

Наступний крок – це перевірка для руху до цільової позиції. Якщо досягнуто цільової позиції, то отримати нову цільову позицію:

```
        void MoveTowardsTarget()
        {
            transform.position =
Vector3.MoveTowards(transform.position, targetPosition,
moveSpeed * Time.deltaTime);
            if (transform.position == targetPosition)        {
                GetNewTargetPosition();        }
        }


        void UpdateAnimation()
        {
            Vector2 movementDirection = (targetPosition -
transform.position).normalized;
        {
            Vector2 movementDirection = (targetPosition -
transform.position);
        {
            Vector2 movementDirection =
        {
            Vector
    // Вектор напрямку руху
            animator.SetFloat(AnimParamX, movementDirection.x);
            animator.SetFloat(AnimParamX, movementDirection.x);
            animator.SetFloat(Anim
            animator
```

Встановити значення параметра анімації для горизонтального руху:

```
            animator.SetFloat(AnimParamY, movementDirection.y);
            animator.SetFloat(AnimParamY, movementDirection.y);
```

Встановити значення параметра анімації для вертикального руху.

ВИСНОВКИ

Roguelike ігри стають все більш популярними серед геймерів, завойовуючи своє місце у сучасній індустрії відеоігор. Їх унікальна комбінація процедурної генерації рівнів, випадкових подій і великої кількості реіграбельності забезпечує глибокий і захоплюючий досвід для гравців.

Взаємодія з оточенням у 2D Roguelike грах відіграє важливу роль, надаючи можливості для дослідження світу гри, виконання завдань, взаємодії з персонажами та впливу на сюжетні лінії. Це розширює можливості геймплею і стимулює гравців до активного дослідження і взаємодії з ігровим світом.

2D Roguelike гри продовжують розвиватися, надаючи геймерам нові механіки, покращену графіку та глибший геймплей. Вони втілюють унікальний жанр і здатні запропонувати незабутні враження та випробування для всіх шанувальників відеоігор.

В результаті роботи було розроблено гру 2D Roguelike з графічним інтерфейсом, який містить всі необхідні компоненти для запуску та геймплею. На першому етапі роботи були розглянуті аналоги, що допомогло поставити основні вимоги до дипломної роботи. Під час проектування надано опис сценарію гри та побудовано діаграму Use-Case, що відображає основні можливості гравця у ігровому просторі.

Під час тестування були перевірені різні аспекти гри, включаючи механіку взаємодії з оточенням, реакцію і штучний інтелект персонажів, процедурну генерацію рівнів та розміщення ресурсів, ефекти візуального та звукового оформлення, а також інші геймплейні елементи. Було виявлено й виправлено помилки, що допомогло створити стабільну та збалансовану гру.

Реалізовано механіку гри та протестовано гру на декількох комп’ютерах. Як подальший розвиток роботи можна розглядати додавання мультиплеєру для можливості гри для декількох користувачів.

ПЕРЕЛІК ВИКОРИСТОВАНИХ ПОСИЛАНЬ

1. Top Reasons Why Computer Games Are Still Popular Among Gamers. URL: https://www.juegostudio.com/blog/top-reasons-why-computer-games-are-still-popular-among-gamers (дата звернення 23.04.2023)
2. What Are Roguelike and Roguelite Video Games? URL: https://www.makeuseof.com/what-are-roguelike-and-roguelite-video-games/ (дата звернення 26.04.2023)
3. Why Are Modern Roguelike Games So Popular? URL: https://gamedevlibrary.com/gamedev-thoughts-why-are-modern-roguelike-games-so-popular-53c1f5a05485 (дата звернення 30.04.2023)
4. Unreal Engine. URL: https://habr.com/ru/hub/unreal_engine/ (дата звернення 14.04.2023)
5. Все, що потрібно для створення гри. URL: https://gamemaker.io (дата звернення 20.04.2023)
6. The Best Gaming Engines You Should Consider for 2023. URL: https://www.incredibuild.com/blog/top-gaming-engines-you-should-consider (дата звернення 20.04.2023)
7. Why Choose Unity 3D for Your Next Game Development Project? URL: https://www.mindinventory.com/blog/unity-3d-game-development/ (дата звернення 04.05.2023)
8. Building A Game With Unity And Blender Free Pdf Book PDF Book. URL: https://freepdf-books.com/building-a-game-with-unity-and-blender-free-pdf-book/ (дата звернення 04.05.2023)
9. C# in 2023: The MOST POPULAR Programming Language? URL: https://www.bytehide.com/blog/c-wants-to-become-the-most-popular-programming-language-in-2022 (дата звернення 10.05.2023)
10. Why should you care about Unreal Engine 5? https://techcrunch.com/2022/04/11/what-is-epic-games-unreal-5/ (дата звернення 10.05.2023)