Національний університет «Полтавська політехніка імені Юрія Кондратюка»
(повне найменування вищого навчального закладу)
Навчально-науковий інститут інформаційних технологій та робототехніки
(повна назва факультету)
Кафедра комп'ютерних та інформаційних технологій і систем
(повна назва кафедри)

Пояснювальна записка
до дипломного проекту (роботи)

магістра
(рівень вищої освіти)
на тему
Розробка комп'ютерної гри у жанрі roguelike

Виконав: студент 6 курсу, групи 601-ТН
спеціальності
122 Комп'ютерні науки
(шифр і назва напряму)
Царьков Р.М.
(прізвище та ініціали)
Керівник Руденко О.А.
(прізвище та ініціали)
Рецензент
(прізвище та ініціали)

Полтава – 2021 року

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ « ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ

КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА
спеціальність 122«Комп'ютерні науки»

на тему

«Розробка комп'ютерної гри у жанрі roguelike»

Студента групи 601-ТН Царьков Ростислав Михайлович

Керівник роботи
кандидат технічних наук
Руденко О.А.

Завідувач кафедри
кандидат технічних наук,
доцент Головко Г.В.

Полтава – 2021

РЕФЕРАТ

Кваліфікаційна робота магістра: 86 с., 55рисунків, 2 додатки, 22 джерела.

Об’єкт дослідження: процес дослідження, вивчення та розробки гри у жанрі roguelike.

Мета роботи:розробка та створення концепту та основних механік гри, з метою подальшої реалізації як повноцінного ігрового продукту.

Методи: збір інформації в галузі створенні комп’ютерних ігор, аналіз жанру roguelike, програмних продуктів, створення макетів та попереднє планування процесу розробки, послідовна реалізація гри.

Ключові слова:комп’ютерна гра, roguelike, ігровий рушій.

ABSTRACT

Master's qualification work: 86 pp., 55 drawings, 2 appendices, 22 sources.

Object of research: the process of research, study and development of the game in roguelike genre.

Purpose: development and creation of the concept and basic mechanics of the game, in order to further implement as a full-fledged game product.

Methods: collection of information in the field of computer games, analysis of roguelike genre, software products, creation of layouts and preliminary planning of the development process, consistent implementation of the game.

Keywords: computer game, roguelike, game engine.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

RPG – Role-Playing Game (рольова гра).

CCG –CollectibleCardGame (колекційна карткова гра).

Сетинг– сукупність різнопланових елементів, які однозначно ідентифікують світ, де відбуваються події певного художнього твору, відеогри або настільної рольової гри.

ШІ–штучний інтелект.

ВСТУП

Комп’ютерні ігри стали справжнім культурним феноменом – виникнувши як нехитрий плід творчої думки програмістів, вони з кожним роком набували все більшої популярності. По всьому світу виросли компанії з розробки ігор, а робота у цій сфері стала рожевою мрією для багатьох умів, які бажають створювати комп’ютерні ігри.

Спроби створити простенькі ігри на цифрових пристроях робилися ще до початку Другої Світової війни, а в 1947 вже було створено першу електронну гру, монітором для якої служив екран військового радара – це був симулятор ворожих ракет. Проте вважається, що першою комп’ютерною грою стала гра "Хрестики нулики", яку зробив А.С. Дуглас 1952 року, з мінімально можливим полем 3х3 клітини. На той момент це було революційним нововведенням. Хрестики нулики – одна з найпопулярніших ігор.

У 1958 році в Нью-Йорку Вільям Хігінботем порадував користувачів новим досвідченим зразком. Ним стала відеогра «Теніс на двох». Відвідувачі його лабораторії могли пограти в теніс на цифровому корті, керуючи своїми ракетками за допомогою джойстиків. Незважаючи на простоту функціоналу, ця гра стала черговим проривом у світі розробки відеоігор.

Проте справжній віртуальний вибух стався 1962 року. Компанія DEC розробила ігровий контролер і разом з комп’ютером PDP-1 стала розповсюджувати як тестову програму гру SpaceWar, яка не мала раніше подібних. Це була перша комп’ютерна гра, що стала по-справжньому популярною.

Комп’ютери на той час були ще громіздкими. Пройшло майже 10 років до того моменту, коли з’явилися компактні плати на транзисторних схемах. У травні 1972 року була представлена MagnavoxOdissey – перша ігрова приставка для телевізора.

З цього моменту ігровий комп'ютерний світ став просуватися семимильними кроками. Розвиток йшов у чотирьох основних напрямках: безпосередньо комп'ютери, телевізійні ігрові приставки, електронні ігрові автомати та кишенькові електронні ігри.

У 1979 році американською компанією MiltonBradley була випущена перша кишенькова ігрова консоль, в яку було вкладено одразу 12 ігор. У 1980 році японська Nintendo, модернізувавши ігри на калькуляторі, здійснила масовий випуск найпростіших монохромних консолей із серією ігор Game&Watch.

З розвитком технологій, віртуальний світ наповнювався звуком, удосконалювалася графіка, додавалася відео. Сьогодні кожен бажаючий може скачати та встановити ігри на будь-який смак.

Процес розвитку комп'ютерних ігор не можна зупинити. Вони стають все більш різноманітними та захоплюючими. На зміну персональним іграм прийшли браузерні ігри, де в онлайн-режимі можна спілкуватися, спільно вирішувати нехай віртуальні, але важливі завдання. Такі ігри, крім розваги, дають змогу освоювати та розвивати навички спілкування, соціалізації, розширювати кругозір. Світ комп'ютерних ігор, як і раніше, дає більше позитивних ефектів, ніж негативних. Важливо лише правильно вибирати собі відповідні ігри і правильно дозувати кількість часу, проведеного біля монітора.

Тема дипломної роботи присвячена створенню комп'ютерної гри у жанрі roguelike. Необхідно провести аналіз існуючих рішень в даному жанрі та доступних програмних продуктів для реалізації відеогри, з метою визначення оптимального варіанту та його подальшої реалізації шляхом створення нового ігрового продукту.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області

Комп’ютерні ігри – це ігри, в які грають або використовують електронні пристрої, такі як ігрові приставки, смартфони, планшети, гарнітури віртуальної реальності або персональні комп’ютери. У них можна грати в Інтернеті, локальних мережах або в автономному режимі. Як і ігри, комп’ютерні ігри дуже різноманітні і включають складні онлайн-світи з кількома гравцями –до простих головоломок для одного гравця. Комп’ютерні ігри різняться за розмірами, включаючи ступінь розвитку розповіді чи характеру, тривалості та складності, жанру чи типу гри [1].

Розробка відеоігор – це процес розробки відеоігор. Зусилля докладає розробник – від однієї особи до міжнародної команди, розпорошеної по всьому світу. Розробка традиційних комерційних ПК та консольних ігор зазвичай фінансується видавцем, і завершення може зайняти кілька років. Інді-ігри зазвичай займають менше часу та грошей, і їх можуть виробляти окремі особи та невеликі розробники. Незалежна індустрія ігор зростає, чому сприяє зростання доступного програмного забезпечення для розробки ігор, такі платформи як Unity та UnrealEngine, а також нових систем розповсюдження в Інтернеті, таких як Steam та Uplay, а також ринку мобільних ігор для Android та пристроїв iOS [2].

На початку 1980-х років після появи перших домашніх комп’ютерів та ігрових приставок один програміст міг керувати майже усіма завданнями, пов’язаними з розробкою гри. Розробка сучасних ігор передбачає наявність широкого кола навичок і персоналу підтримки. Для роботи над одним проектом

потрібні цілі команди, до складу яких зазвичай входять представники ряду спеціалізацій:

- Продюсер – людину з цією посадою можна також називати менеджером проекту, керівником проекту або директором.
- Видавець комп'ютерних ігор – компанія, яка публікує / видає комп'ютерні ігри.
- Команда розробки, в яку входять наступні посади:
  - Геймдизайнер– це людина, яка проектує ігровий процес, задумуючи і проектуючи правила і структуру гри.
  - Художник малює те, як буде виглядати гра.
  - Ігровий програміст – це інженер-програміст, який в основному розробляє комп'ютерні ігри або відноситься до них програмне забезпечення (таке як інструменти розробки ігор).
  - Геймдизайнер рівнів – це людина, яка створює рівні, завдання або місії для комп'ютерних відеоігор, використовуючи при цьому інструменти розробки ігор або інші програми
  - Звукорежисери є технічними фахівцями, відповідальними за звукові ефекти і звуковий супровід протягом всієї гри.
  - Тестувальник аналізує комп'ютерну гру і документально фіксує знайдені ним дефекти і помилки, що є частиною всього процесу контролю якості[2].

Процес розробки гри зазвичай включає наступні етапи:

- підготовка;
- уточнення геймдизайну;
- виробництво;
- підтримка.

Етапи можуть змінюватися в залежності від уподобань фірми і особливостей проекту.

1.2 Жанр roguelike

Roguelike (буквально «rogue-подібні») –жанр комп'ютерних ігор. Характерними особливостями класичного roguelike є генеруються випадковим чином рівні, поступовість дій і незворотність смерті персонажа – в разі його загибелі гравець не може завантажити гру і повинен почати її заново. Багато roguelike виконані в декораціях епічного фентезі під сильним впливом настільних рольових ігор на зразок Dungeons&Dragons [3].

Жанр сходить до гри 1980 року Rogue (рис. 1.1). Хоча і до неї виходили подібні ігри, такі як BeneathAppleManor (рис. 1.2), саме Rogue стала зразком для незліченних наслідувань. Подібні ігри, що розповсюджувалися в вигляді відкритого коду, були вкрай популярні в середовищі американських програмістів і студентів в 1980-х - 1990-х роках. При наявності безлічі варіантів і відмінностей між окремими іграми основні принципи ігрового процесу roguelike залишалися незмінними. У 2008 році ці консервативні принципи були описані в рамках так званої «Берлінської інтерпретації» [3].

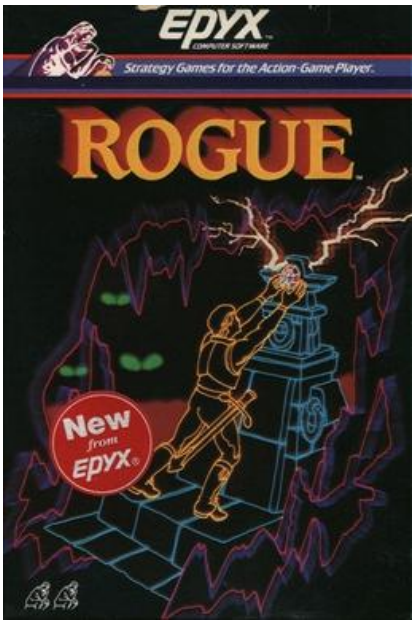

Рисунок 1.1 – Обкладинка гриRogue

Рисунок 1.2 – Обкладинка гри BeneathAppleManor

У 2000-х роках почали з’являтися ігри, які включають в себе один або кілька основних принципів roguelike разом з елементами ігор інших жанрів. Подібні ігри іноді класифікують як roguelike-like («Rogue-подібно-подібні») і roguelite («злегкаrogue») [4].

Жанр названий по імені прабатька – гри Rogue, розробленої для системи UNIX в 1980 році, хоча вийшли раніше гри BeneathAppleManor і SwordofFargoal також відносять до жанру roguelike. У 1980-х і 1990-х роках roguelike придбали величезну популярність серед програмістів і студентів американських університетів, що призвело до появи великої кількості наслідувань і відгалужень, які зберігали, проте, характерні особливості ігор-попередниць. До числа найбільш відомих roguelike належать Hack, NetHack, AncientDomainsofMystery (рис 1.3), Moria, Angband і TalesofMaj'Eyal (рис. 1.4), а також японська серія комерційних ігор MysteryDungeon [4].

Рисунок 1.3 – Обкладинка гри AncientDomainsofMystery

Рисунок 1.4 – Обкладинка гри TalesofMaj’Eyal

Ігрова система зазвичай є адаптацією систем, використовуваних в настільних рольових іграх, таких як Dungeons&Dragons або GURPS. Карти, як правило, генеруються випадковим чином при кожному новому запуску гри. Більшість ігор (крім розрахованих на багато користувачів) є покроковими, без обмеження часу [4].

Для зображення ігрових об’єктів в традиційних roguelike-іграх використовуються символи ASCII (рис. 1.5). Так, персонаж зазвичай позначається символом «@», монстри (зазвичай ворожі) – буквами алфавіту, гроші – символом «$», їжа – символом «%» і т.д. Деякі ігри використовують замість ASCII-символів картинки фіксованого розміру. Дії персонажа управляються з клавіатури в стилі, трохи нагадує редактор vi. Більшість команд викликаються натисканням однієї-двох клавіш, наприклад, «r» (read), щоб прочитати сувій або книгу, «e» (eat), щоб що-небудь з’їсти і т.п [5].

Рисунок 1.5 – Кадр з гри NetHack

Нове звернення до жанру roguelike сталося в 2000-і роки з розквітом розробки інді-ігор і появи «roguelike-подібних» ігор з нечіткою жанровою приналежністю. Такі ігри виглядали не власне roguelike в класичному розумінні цього терміна, але з'єднання елементів roguelike та ігор інших жанрів. До числа ранніх прикладів roguelike-подібних ігор відносяться StrangeAdventuresinInfiniteSpace (2002) (рис. 1.6) і її продовження WeirdWorlds: ReturntoInfiniteSpace (2005), розроблені студією DigitalEel– обидві гри були присвячені дослідженню космосу, включали в себе випадкову генерацію планет і зустрічей з різними небезпеками і необоротну смерть. Джерелом натхнення для обох ігор послужили одночасно пригодницька гра Starflight і гри-roguelike типу Nethack [5].

Рисунок 1.6 – Обкладинка гри StrangeAdventuresinInfiniteSpace

З розвитком комп'ютерів і ігрової індустрії стався відхід від класичного визначення «рогаликів». Нову формацію «roguelike» по англ. називають «roguelike-like» («подібні rogue-подібним»). Такі ігри найчастіше мають випадково генеруються рівні і перманентну смерть, але без покрокового режиму, і основний жанр може бути – екшен або платформер. Гравці спокійно поставилися до такого видозміни класичного жанру, більше звертаючи уваги - приносить гра задоволення чи ні [5].

З появою більш потужних комп'ютерів та ігрових консолей з'явилися нові ігри, що зберігають основні принципи roguelike– в першу чергу процедурне створення рівнів і незворотність смерті ігрового персонажа - але включають в себе елементи ігор інших жанрів, незвичні для roguelike теми і стилі графіки. Популярність таких інді-ігор, як Spelunky (рис. 1.7), TheBindingofIsaac, FTL: FasterThanLight і RogueLegacy (рис. 1.8) сприяла поширенню принципів roguelike в ігри інших жанрів. Для відмінності між традиційними roguelike і іграми змішаних жанрів, які використовують лише деякі елементи класичної концепції, іноді застосовують терміни «roguelike-like» («Rogue-подібно-

подібні») і «roguelite» («легкі» roguelike). Інші ігри, такі як Diablo і UnRealWorld– важливі віхи в еволюції жанрів Action / RPG і симуляторів виживання відповідно - створювалися під впливом roguelike [5].

Рисунок 1.7 – Обкладинка гри Spelunky

Рисунок 1.8 – Обкладинка гри RogueLegacy

1.3 Огляд відеоігор в жанріroguelike

1. DeadCells– комп'ютерна інді-гра в змішаному жанрі roguelike і метроідванії, розроблена і випущена французькою студією MotionTwin для платформ Windows, macOS, Linux, ігрових консолей NintendoSwitch, PlayStation 4, і Xbox One в 2018 році; раніше протягом року попередні версії гри пропонувалися гравцям через систему раннього доступу. Пізніше гра була також перенесена на iOS і Android. На рисунку 1.9 зображена обкладинка гри [6].

Рисунок 1.9 – Обкладинка гри DeadCells

В ході DeadCells гравець управляє істотою, що намагається вибратися з лабіринту. Рівні гри, виконані в дусі двомірного платформера, генеруються процедурним чином; по ним розкидані вороги і різні скарби, в тому числі зброю зі випадково генеруються характеристиками. Подібно до ігор в жанрі roguelike, у персонажа DeadCells тільки одна «життя» – якщо він загине, гравець буде змушений почати гру з самого початку. Деякі одного разу отримані здібності, що відкривають доступ до нових, раніше недоступних областях гри, переносяться і в наступні проходження [7].

Жанр DeadCells описується як поєднання жанрів roguelike з їх процедурно генеруються рівнями і метроідваній з їх action-геймплеєм і зв'язковими світами, які вимагають поступового дослідження. Істота, яким керує гравець, являє собою розумний згусток клітин, в кожному новому проходженні заволодіває трупом одного зі страчених в'язнів у величезній в'язниці. В ході гри персонаж обстежує різні підземелля, перемагає ворогів які їх населяють і збирає численні предмети. Персонаж може носити з собою дві зброї і два допоміжних предмета і знаходити в підземеллях нові – від мечів і метальних ножів до гранат, капканів та турельних установок. Зброя і предмети мають різні випадково генеруються характеристиками з додатковими ефектами. Так, зброя може наносити підвищений шкоди спливаючим кров'ю ворогів або змушувати тіла переможенийпротивників вибухати. Випадають з переможених ворогів предмети – «клітки» – дозволяють в кінці кожного рівня купувати постійні бонуси, як, наприклад, можливість більше разів використовувати склянку для відновлення здоров'я або особливо потужну зброю по знайденим в підземеллях кресленнями. Клітини можна витратити тільки в кінці рівня; якщо ігровий персонаж загине, чи не дістанеться до виходу з рівня, він втратить всі зібрані клітини. На рис. 1.10 зображений кадр з гри [7].

Кожен рівень в кожному проходженні генерується випадковим чином – при цьому гра збирає заплутані лабіринти з заздалегідь визначених елементів, випадково розкидаючи по ним ворогів і предмети. Гра проводить гравця через битви з ворогами зі складною поведінкою і передбачає, що персонаж буде часто гинути, а гравець – вчитися на своїх помилках. Хоча більшості битв в грі можна уникнути, в ході проходження DeadCells гравець повинен перемогти декількох особливо складних босів [7].

Рисунок 1.10 – Кадр з гри DeadCells

2. DwarfFortress (з англ. –«Фортеця дварфів»)– комп’ютерна гра, що поєднує в собі елементи симулятора будівництва та управління і roguelike, що розробляється братами Тарном і Заком Адамсами з 2002 року. Гра є безкоштовною і існує виключно на пожертви прихильників; перша альфа-версія гри була випущена в 2006 році. На 2021 рік гра все ще знаходиться на стадії альфа-версії, незважаючи на регулярний випуск оновлень. Логотип гри на рис. 1.11[8].

Рисунок 1.11 – Логотип гри DwarfFortress

В основному режимі гравець непрямим чином керує групою дварфів, намагаючись побудувати підземну фортецю в процедурно створеному світі; протягом ігрової сесії гравець повинен планувати будівництво нових споруд, організовувати побут жителів фортеці, організовувати її захист від ворогів і займатися подібними справами. У DwarfFortress немає будь-якої кінцевої мети,

перемога в грі не передбачена. Девіз гри – «Losingisfun» («Програвати весело»). Додатковий режим Adventure більше нагадує традиційні roguelike. Незважаючи на навмисно спрощену графіку і управління з клавіатури, DwarfFortress примітна надзвичайно складною опрацюванням вигаданого світу і симуляцією безлічі його складових [8].

DwarfFortress включає в себе три ігрових режиму – всі вони розгортаються в процедурно створеному світі, причому повна генерація світу відбувається з початком кожної гри. Основний режим «Фортеця» (англ. Fortress) являє собою симулятор будівництва та управління, в якому гравець, керуючи групою дварфов, повинен побудувати підземну фортецю і підтримувати її якомога довше, перетворивши в процвітаючу цивілізацію. У режимі «Пригод» (англ. Adventure) гравцеві пропонується досліджувати той же самий світ, керуючи одним персонажем, як в більш традиційних roguelike; при цьому гравець може знайти в світі гри і занедбані фортеці, побудовані ним самим в режимі «Фортеця». Режим «Легенди» (англ. Legends) являє собою віртуальну книгу, що включає в себе різні історії і опису того ж самого світу; крім іншого, в цьому режимі гравець може перечитати і історію фортець, збудованих в режимі «Фортеця» [9].

Першим кроком в DwarfFortress є генерація світу; за раз в одному створеному світі можна вести тільки одну гру. Гравець може поставити ряд параметрів, в тому числі розмір світу; його ворожість, тобто кількість диких областей з небезпечними ворогами; багатство корисними копалинами і тривалість історії до початку гри. Чим довше задана історія в роках, тим більше об’єктів і інформації з минулого - міфів, легенд, руїн – зустріне гравець. Різні символи на карті позначають генеруються дороги, пагорби, міста різних цивілізацій, і вони змінюються в міру процесу генерації, показуючи, як еволюціонує створюваний світ [9].

Живі істоти (а також представлена в грі нежить) складаються з окремих частин тіла і внутрішніх органів, для яких індивідуально розраховуються пошкодження в ході бою, падіння з висоти, попадання під обвал або опіку. Пораненим героям потрібно не абстрактна «аптечка», як це прийнято в більшості ігор, а промивання рани, накладення швів і, в разі перелому, фіксація пошкодженої кінцівки. Кадр з гри DwarfFortress на рис. 1.12 [9].

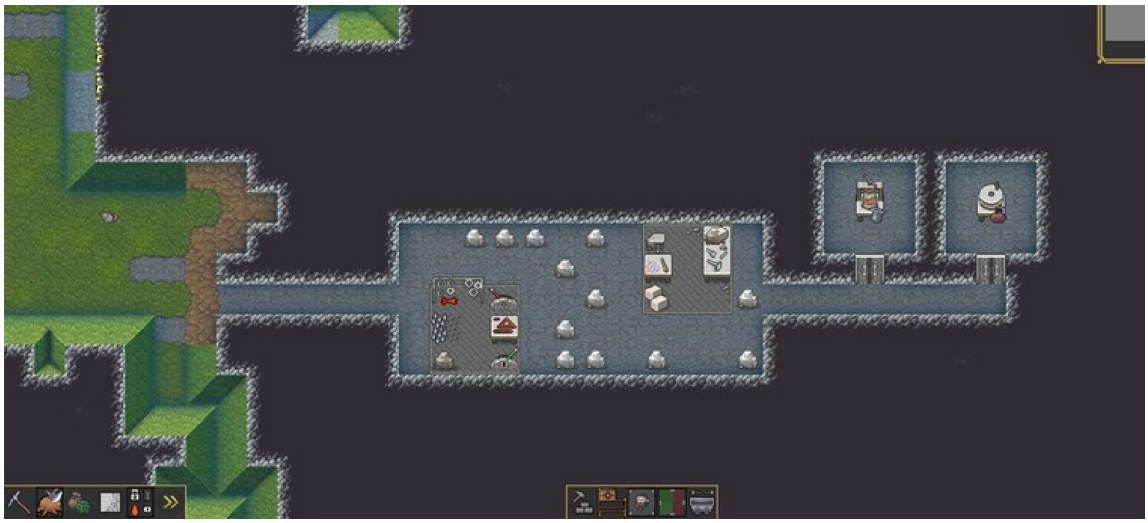

Рисунок 1.12 – Кадр з гри DwarfFortress

Шари гірських порід утворені десятками різних мінералів, причому дотримується певний порядок: зверху може бути грунт, під нею глина, ще нижче йдуть осадові породи; в деяких місцях виявляються водоносні шари, через які не можна прокладати тунелі без попередньої відкачки води насосами, заморожування або споруди кессонів. Використовуючи реалізовані в грі механізми (водяне колесо, насоси, затвори і натискні плити) можнареалізувати машину Тюрінга, калькулятор і спрощену версію гри SpaceInvaders [9].

У грі присутні кілька десятків видів рослин, які можна вирощувати на фермах – в тому числі і підземних (під землею, наприклад, ростуть вигадані кар’єрники). Частина рослин йде в їжу, частина вимагає додаткової переробки і застосовується в текстильній промисловості або пивоварному виробництві. Всі гриби в DwarfFortress є вигаданими, ростуть і розмножуються під землею [9].

Поряд з економічною і геологічною складової гра відрізняється моделюванням психологічного клімату в підопічному гравцеві колективі. Кожен з дварфів має власні риси характеру, родинні зв'язки і в разі втрати близьких або незадоволення основних потреб (відсутність житла, погане харчування) можлива депресія або спалахи агресії, які проявляються або у вандалізмі, або в нападі на одноплемінників. Залежно від характеру дварфа, різні речі викликають позитивні / негативні думки і впливають на його життя. Крім того, іноді дварфи стають жертвами нав'язливої ідеї по створенню будь-якої речі, і в такому випадку єдиний шанс зберегти їх душевне здоров'я – дати їм все необхідне для роботи [9].

3. DarkestDungeon (укр. Найтемніше підземелля) – відеогра, roguelike /dungeoncrawl, створена командою інді-розробників RedHookStudios. Гра була випущена у ранньому доступі в січні 2015 року та повністю 19 січня 2016 року для Microsoft Windows і OS X, пізніше також для PlayStation 4, PlayStationVita, iPad, NintendoSwitch і Xbox One. Обкладинка гри на рис. 1.13 [10].

Рисунок 1.13 – Обкладинка гри DarkestDungeon

Гравець управляє групою шукачів пригод, обстежують різні місця навколо старовинного маєтку і під ним. Перед кожним походом гравець

повинен вибрати чотирьох героїв з запропонованого набору і провести їх через генеруються випадковим чином (не завжди) підземелля, збираючи скарби, родинні реліквії і знищуючи зустрічаються на шляху різних чудовиськ. Перед гравцем стоїть завдання зберегти життя і психічне здоров'я героїв. Рівень психологічної напруги (стресу) росте кожен раз, коли герої отримують удари від ворогів, потрапляють в пастки, взаємодіють з деякими диковинками або відступають з поля бою; якщо у бійця здають нерви, він починає вести себе неадекватно і може самовільно виконати будь-яку дію в бою, позбавляючи гравця можливості віддати наказ. По ходу дослідження підземелля персонажі набувають різні риси характеру – постійні особливості поведінки: як негативні (алкоголізм, клептоманія), так і позитивні, які роблять героя більш хоробрим і витривалим. Вороги в DarkestDungeon належать до певних класів або типів. Деякі риси характеру героїв пов'язані з класами противника. Позитивні риси характеру можуть давати герою бонуси (наприклад, збільшення шкоди, точності або шансу нанесення критичного шкоди), коли той стикається з ворогами певних класів. Тим часом, негативні риси можуть накладати негативні ефекти (наприклад, підвищений стрес) при зіткненні з ворогами певних класів. Крім того, є також певні навички героя і артефакти, що дають бонуси проти деяких ворожих класів. У грі 17 різних класів персонажів: Abomination, Antiquarian, Arbalest, BountyHunter, Crusader, GraveRobber, Hellion, Highwayman, Houndmaster, Jester, Leper, Man-at-Arms, Occultist, PlagueDoctor, Vestal,Flagellant і Sheildbreaker. У кожного класу унікальні навички і поведінку. Кадр з гри на рис. 1.14 [10-11].

Рисунок 1.14– Кадр з гри DarkestDungeon

4. RogueLegacy– комп’ютерна гра в жанрі платформера з елементами roguelike. Гравець досліджує процедурно згенеровані локації, видобуває золото, зброю і броню для свого персонажа, а також бореться з ворогами. В RogueLegacy присутні чотири локації – замок, ліс, вежа і підземелля. У кожній з них свій рівень складності, свої вороги і свій бос. Мета гравця – перемогти чотирьох босів в різних локаціях, після чого гравець отримує можливість битися з п’ятим, фінальним босом. Обкладинка гри на рис. 1.15 [12].

У разі загибелі персонажа гравець повинен вибрати одного з трьох нових персонажів – дітей загиблого героя; в черговому проходженні гравець управляє вже цим спадкоємцем. У кожного з героїв є свій клас, заклинання, яке він може використовувати, а також свої особливості. Наприклад, персонаж може страждати хворобою Альцгеймера, СДУГ, недоумством, мати нетрадиційну сексуальну орієнтацію і так далі [12].

Рисунок 1.15 – Обкладинка гри RogueLegacy

Після загибелі персонажа гравець може використовувати зібране ним золото для поліпшення характеристик майбутніх персонажів, а також витратити на покупку нової зброї, броні і рун, кожна з яких надає персонажу певнівміння. Щоб отримати доступ до певної руно, гравцеві необхідно розблокувати її - для цього потрібно знайти в грі кімнату з магічним скринею і виконати певне завдання, наприклад, дійти до скрині без єдиного стрибка або знищити всіх ворогів в кімнаті. Якщо умова виконана успішно, персонаж зможе відкрити магічний скриню і активувати знайдену руну, що дозволяє персонажеві, наприклад, здійснювати подвійний стрибок. Крім рун, по ходу гри можна виявити скрині з кресленнями, відкривши які, гравець отримує доступ до меча або броні, які згодом гравець зможе надіти на свого персонажа. Зброя і броня підвищують атаку, захист і ману гравця, а деякі елементи броні і мечі можуть дати гравцеві ті чи інші здібності, такі як підвищення шансу критичного удару або відновлення очок здоров’я після вбивства ворога. Кадр з гри на рис. 1.16 [12-13].

Рисунок 1.16– Кадр з гри RogueLegacy

Перед тим як персонаж увійде в замок, він може скористатися послугами одного з трьох персонажів, що стоять біля входу в замок. Серед них коваль, чарівниця і архітектор. У коваля гравець може надягати на свого персонажа мечі, шоломи, нагрудники, поножі з рукавичками і плащі; у чарівниці гравець може повісити на свого персонажа руни, які надають персонажеві додаткових можливостей; а архітектор може відновити ігрову карту, яку гравець досліджував в минулий раз, забираючи в якості плати за це 40% золота, якого гравець добуде при наступному запуску рівня [13].

5. LoopHero (з англ. – «герой циклу») – комп’ютерна гра в жанрі roguelike з непрямим управлінням, розроблена російською студією FourQuarters і випущена компанією DevolverDigital 4 березня 2021 року на персональних комп’ютерах (Windows, macOS, Linux). За сюжетом гри, світ був поміщений в тимчасову петлю, пам’ять його мешканців була стерта, а більша частина світу занурилася в пітьму, яка розсіюється, коли люди згадують про те, що за нею знаходилося. Обкладинка гри на рис. 1.17 [14].

Рисунок 1.17 – Обкладинка гри LoopHero

LoopHero є грою жанру roguelike з елементами RPG, KCG і містобудівного симулятора. Ігровий світ являє собою зациклену стежку, на якій знаходиться вогнище. Протагоніст самостійно йде по стежці і бореться зі що зустрічаються монстрами, при цьому у гравця немає можливості впливати на його дії. Кожен раз, коли герой проходить повз багаття, він лікується. З кожним колом вороги стають сильнішими. Перед битвою з розумними противниками протагоніст вступає з ними в діалоги, через які гравцеві передається інформація про світ гри і пояснюються ігрові механіки [14].

Убиті монстри приносять герою досвід, який гравець в подальшому може витратити на навички персонажа, також з них випадає екіпірування для героя, яку гравець може міняти в будь-який момент, і карти місцевості - наприклад, гори, поля, річки, ліси, храми. Карти місцевості використовуються для того, щоб розширювати ігрову зону, при цьому карти різного типу по-різному впливають на героя: так, село буде лікувати головного героя і видавати завдання, а ліси будуть заважати герою, виставляючи на його шляху щурововків. Також карти взаємодіють один з одним: так, якщо поставити поруч з селом особняки вампірів, село буде захоплена упирями, але якщо герой тричі проб’ється через неї з боєм, вона перетвориться в посилену село і буде давати

більше зцілення і просунуті завдання. У грі не представлено списку поєднуваних клітин, що залишає гравцеві поле для експериментів, однак про деякі з них можна дізнаватися з діалогів персонажів. Після того як карта розростеться до певної межі, вогнище зміниться на лігво боса, перемога над яким призведе гравця на наступний акт історії. Кадр з гри на рис. 1.18 [15].

Рисунок 1.18– Кадр з гри LoopHero

Комбінації локацій, заповнення світу і вбивство монстрів приносять ресурси і магічні сфери сутностей. Поки герой знаходиться біля багаття, можна припинити забіг, отримавши всі ресурси; в будь-який момент гравець може змусити героя відступити, що також завершить забіг і принесе 60% зароблених ресурсів; а в разі смерті героя, гравець отримає тільки 30% зібраної сировини. По завершенні забігу ці ресурси можна витратити на поліпшення табору героя, що дасть йому постійні бонуси на всі наступні партії. Ресурси можна переводити в інші види сировини за допомогою алхімії, при цьому створювати можна тільки ті їх види, які були раніше отримані гравцем і розпорошені. Також в таборі можна міняти клас героя; всього в грі є три класи – стандартний воїн, шахрай і некромант. Кожен клас має унікальний ігровий процес: так, шахрай

замість екіпіровки отримує трофеї, які можна обміняти по завершенні кожного кола, а некромант піднімає з мертвих скелетів, які борються на його стороні, і предмети екіпіровки впливають не на самого некроманта, а на скелетів [15].

Ознайомившись з популярними представниками жанру roguelike можна виділитиспільні особливості та особливості кожної відеогри, які роблять їх унікальними.

До спільного можна віднести:

- Випадкова генерація будь-чого. Чи то рівень, чи то характеристики предмету, саме через випадковість кожне проходження не схоже на попереднє.
- «Одне» життя. Програш стимулює гравця до експериментів в наступних проходженнях.
- Прогрес проходження скидається.

Особливості DeadCells:

- Великий світ, який хочеться досліджувати.
- Система розвитку, яка не скидається при смерті гравця.
- Унікальна поведінка ворогів.

Особливості DwarfFortress:

- Широкий вибір настройок складності.
- Деталізація у всіх аспектах світу.

Особливості DarkestDungeon:

- Покрокові бої, що робить гру більш тактичною.
- Найманців, кожен з яких зі своїм характером.
- Розвиток міста за отриману в підземеллях валюту.

Особливості RogueLegacy:

- Система нащадків, які отримують частину ефектів від пращура.
- Скрині, які впливають на розвиток ігрового персонажу.

Особливості LoopHero:

- Гравець не керує героєм.
- Здатність впливати на складність проходження.

1.4 Огляд ігрових рушіїв

Unity. Unity– це кросплатформовий ігровий рушій для розробки 2D та 3D ігор будь-якого жанру та формату.

Цей двигун виграє за рахунок своєї широкої бази знань, величезної кількості прикладів і шаблонів, великої спільноти та низького порога входу.

Приклад роботи в Unity на рис. 1.19.

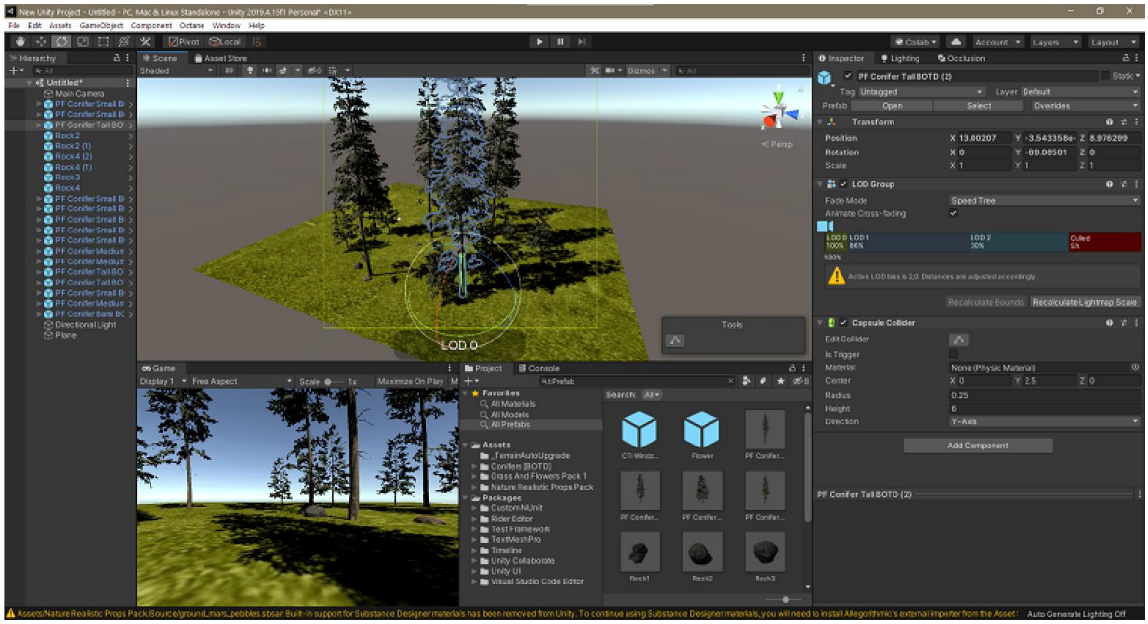

Рисунок 1.19 –Приклад роботи в Unity

Мова програмування: C#, NoCode (Bolt).

Вартість: Безкоштовно та платні тарифи для великих команд.

Плюси Unity:

- гнучкий і розширюваний двигун. Багато компонентів для роботи з коробки;
- велика кількість шаблонів та прикладів робіт;
- велика кількість уроків та курсів для початківців;

- величезна база знань, велика спільнота;
- простота у створенні з допомогою C# і Bolt;
- величезна кількість успішних кейсів;
- кросплатформний.

Мінуси Unity:

- для розробки оптимальної гри необхідно глибоко зануритися в нюанси двигуна;
- багато сирих версій, безліч багів;
- відсутність відкритого коду для невеликих команд розробників;
- величезний розмір;
- у пріоритеті розробки оновлень рушія – мобільні платформи;
- рушій створений насамперед для бізнесу. Тому його модель підтримки орієнтована на бізнес, а не на зручність для розробників [16].

UnrealEngine.Наступний монструозний ігровий рушій – це UnrealEngine. Його історія триває десятиліттями, проте зараз – він здається досяг своєї величі. Рушій розробляється компанією EpicGames і є передовим рішенням створення великих AAA-ігор.

Приклад роботи в UnrealEngine на рис. 1.20.

Мова програмування: C++, NoCode (Blueprints).

Вартість: Безкоштовно (з умовами).Плюси UnrealEngine:

- потужний редактор на всі випадки життя;
- гнучка архітектура ігрового рушія;
- ігровий рушій розробляється навіть для ігор автора. Тому він насамперед націлений на інших розробників, а не бізнес, як у випадку з Unity.
- готовий до AAA-проектів із «коробки»;
- кросплатформний;

Мінуси UnrealEngine:

- високий поріг входу;
- більш закрита і не така численна спільнота;
- акцент – на ААА-проекти;
- розмір рушію та його вимогливість [16].

Рисунок 1.20 – Приклад роботи в UnrealEngine

Stride. Близький за духом Unity ігровий рушій з відкритим вихідним кодом. Його розробка почалася нещодавно (раніше він називався Xenko), проте він буде хорошим вибором для тих - кому важлива безплатність і відкритість вихідного коду.

Приклад роботи в Stride на рис. 1.21.

Мова програмування: C#.

Вартість: Безкоштовно (OpenSource).Плюси Stride:

- відкритий вихідний код;
- нижчий поріг входу, ніж у UnrealEngine;
- схожість з архітектури та інструментів із вищезгаданим Unity;
- C# 9 та актуальні технології, порівняно з Unity;
- потужна підтримка VR.

Мінуси Stride:

- рушій та спільнота ще молода;
- не така велика кількість прикладів та навчальних матеріалів;
- вогкість деяких компонентів;
- більш бідна кросплатформа.

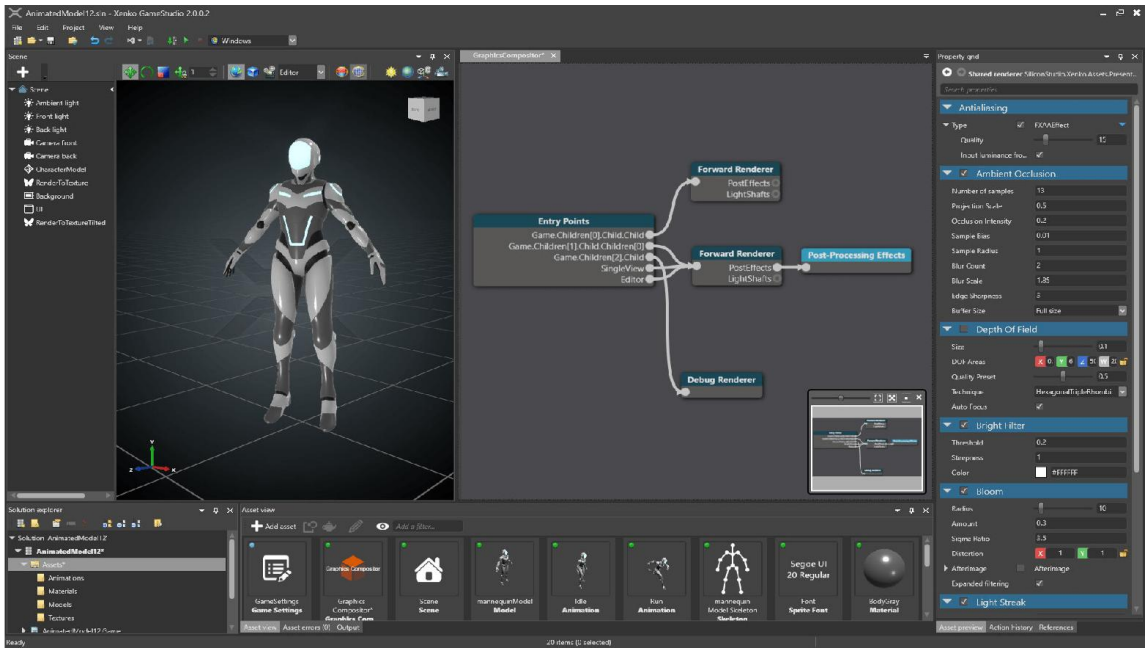

Рисунок 1.21 – Приклад роботи в Stride

GDevelop. Потужний конструктор ігор з відкритим кодом для створення крос-платформних 2D ігор будь-якої складності без знання програмування.

Приклад роботи в GDevelop на рис. 1.22.

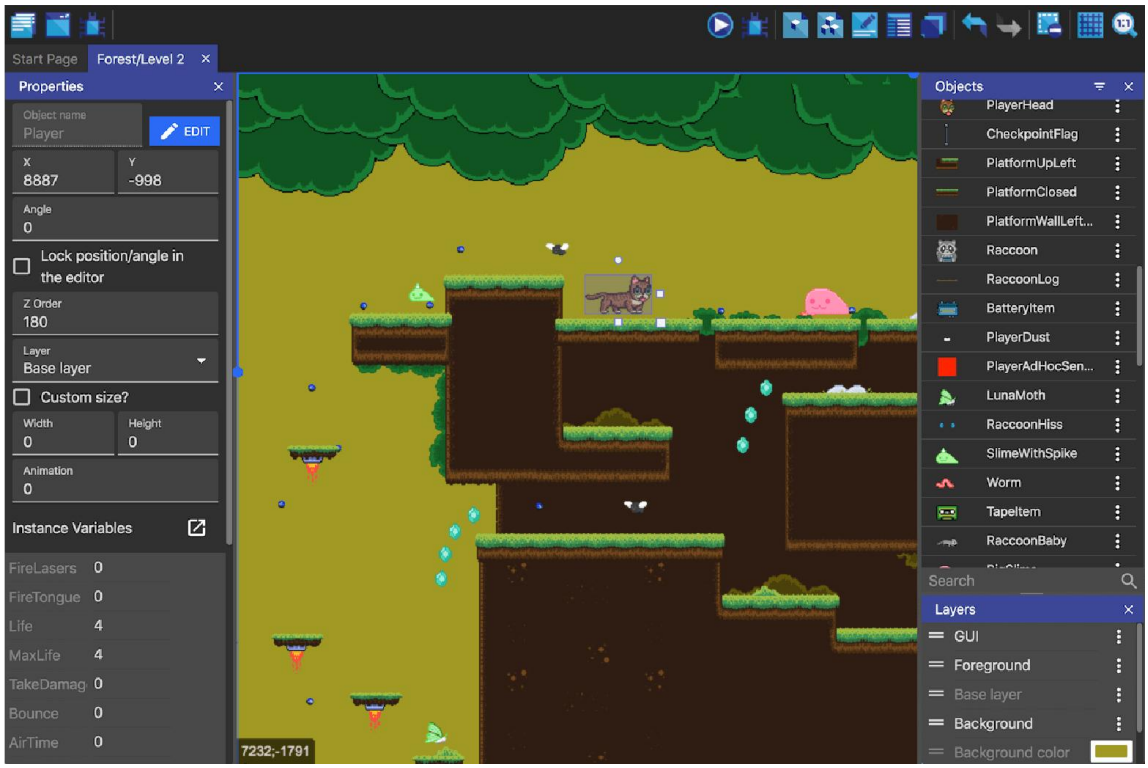

Рисунок 1.22 – Приклад роботи в GDevelop

Мова програмування: не потрібна. На основі подій.

Вартість: Безкоштовно (Open-Source).Плюси GDevelop:

- розробка крос-платформних ігор без програмування;
- гнучкий та розширюваний редактор;
- безкоштовний з відкритим кодом;
- підтримка шейдерів.

Мінуси GDevelop:

- не найбільше спільнота;
- мало прикладів;
- не найзрозуміліший інтерфейс зі старту;
- обмеженість 2D іграми.

Defold. Один з ігрових рушіїв, що набирають популярність, для розробки 2D (і в деяких випадках 3D) проектів. Рушій безкоштовний і підтримує безліч платформ.

Приклад роботи в Defoldна рис. 1.23.

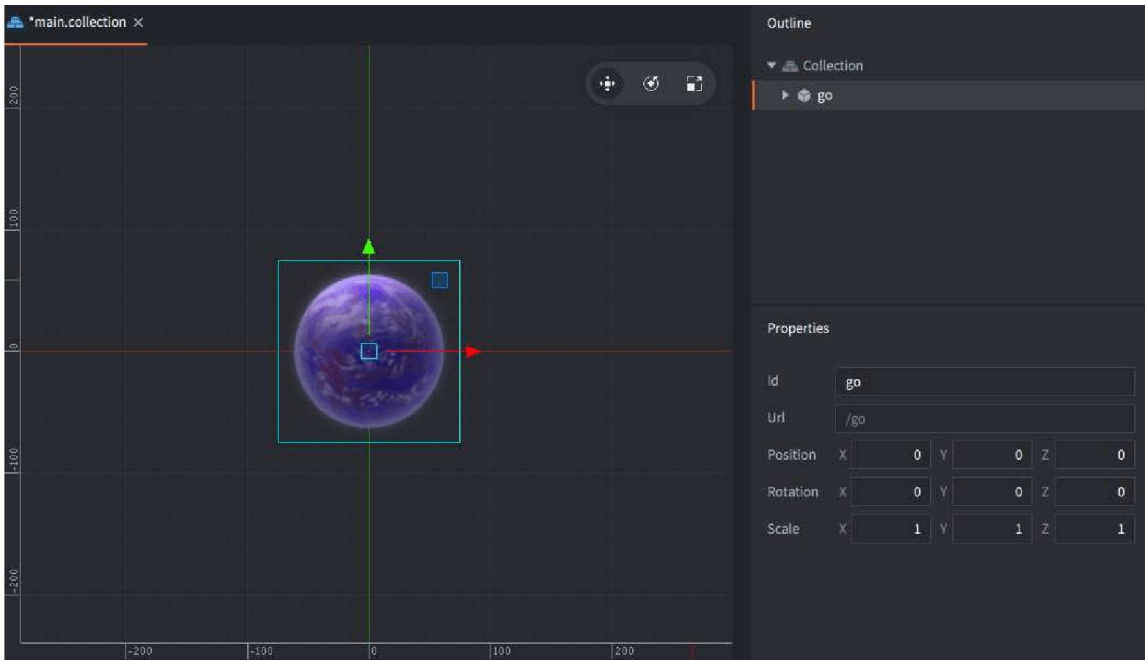

Рисунок 1.23 – Приклад роботи в Defold

Мова програмування: Lua (верхній рівень), C++ (нижній рівень) та інші (розширення рушія).

Вартість: Безкоштовно.Плюси Defold:

- безкоштовний двигун;
- кросплатформний;
- потужні інструменти;
- низький поріг входу та гарна документація.

Мінуси Defold:

- не найприємніший інтерфейс, до якого потрібно звикнути;
- не дуже велика (поки що) спільнота;
- у першу чергу для 2D.

Construct 3. Простий, зручний та гнучкий конструктор ігор, з підтримкою кросплатформи, що дозволяє створювати 2D ігри будь-якого жанру без знань програмування.

Приклад роботи в Construct 3 на рис. 1.24.

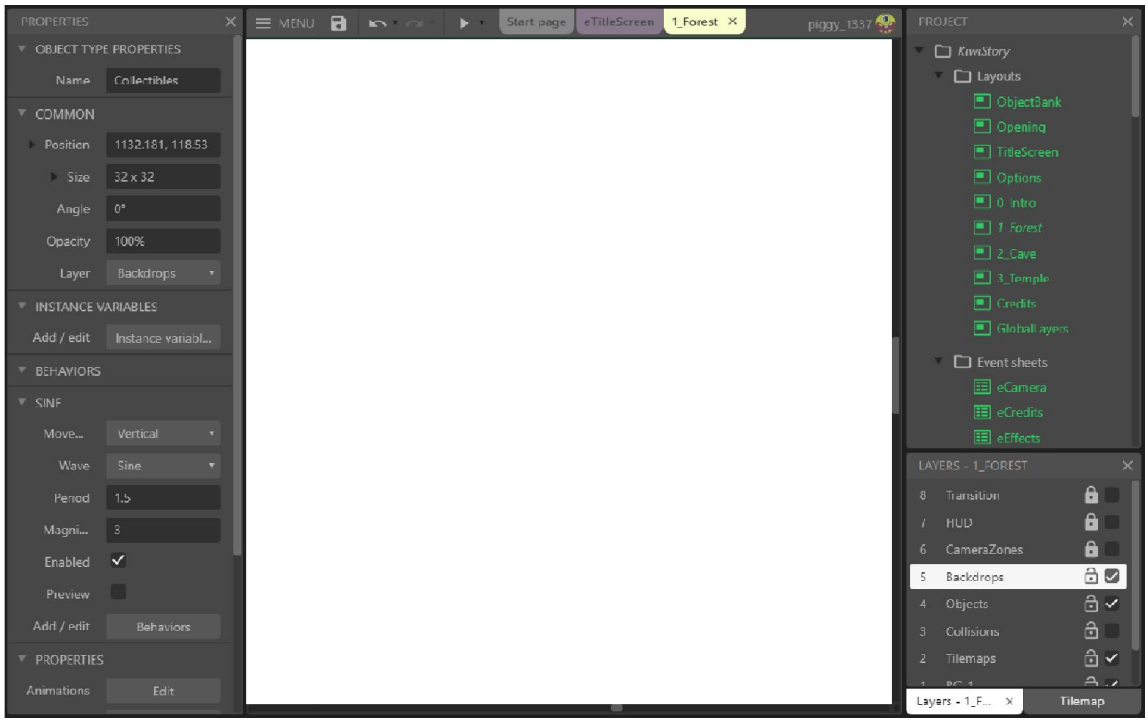

Рисунок 1.24 – Приклад роботи в Construct 3

Мова програмування: Не потрібна. На основі подій.

Вартість: Безкоштовно (з сильними обмеженнями), повний набір з місячною підпискою.Плюси Construct 3:

- не вимагає програмування;
- велика кількість плагінів та розширень;
- зручний та простий інтерфейс, доступний з будь-якого пристрою із браузера;
- крос-платформний;
- велика спільнота, багато прикладів;

Мінуси Construct 3:

- ціна може бути велика;
- акцент на 2D-проекти;
- не найкраща оптимізація;
- для повноцінної роботи потребує підключення до інтернету[16].

GameMakerStudio 2. Ще один представник конструкторів ігор – GameMakerStudio 2. Включає в себе безлічфункцій, шаблонів і прикладів для ігор.

Приклад роботи в GameMakerStudio 2 на рис. 1.25.

Рисунок 1.25 – Приклад роботи в GameMakerStudio

Мова програмування: не потрібна, але підтримує скрипти.

Вартість: Безкоштовно (з обмеженнями), повний набір за місячну підписку.Плюси GameMakerStudio:

- не потребує програмування;
- потужні інструменти;
- велика кількість шаблонів та прикладів;
- велика спільнота.

Мінуси GameMakerStudio:

- не найоптимізованіший рушій;
- у деяких моментах досить недороблений [16].

CryEngine. Рушій розвивається дуже давно, володіє потужним інструментарієм та підтримкою кросплатформи.

Приклад роботи в CryEngine на рисунку 1.26.

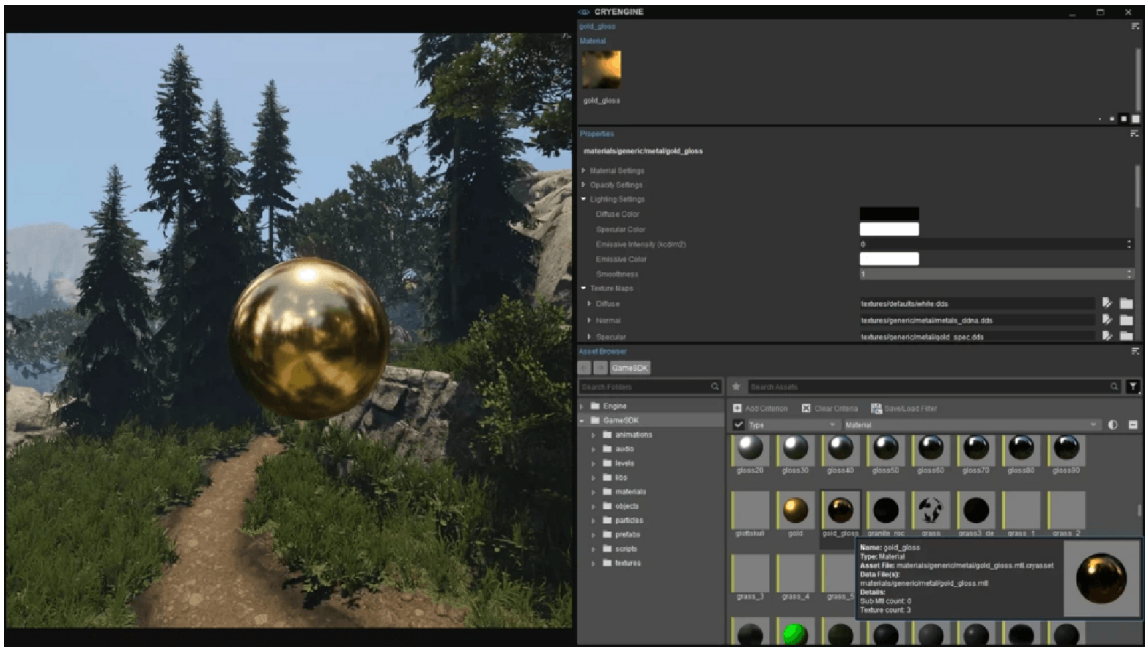

Рисунок 1.26 – Приклад роботи в CryEngine

Мова програмування: C#, C++, FlowGraph.

Вартість: Безкоштовно (з обмеженнями).Плюси CryEngine:

- потужний ігровий редактор-пісочниця;
- величезний потенціал для AAA-проектів;
- один із найпотужніших рендерів;
- гарна документація.

Мінуси CryEngine:

- сильний наголос на розробку шутерів;
- не найзручніший інструментарій;
- для великих проєктів потрібні серйозні знання в C++ [16].

РОЗДІЛ 2

ПРОЄКТУВАННЯ ВІДЕОГРИ

2.1 Проєктні рішення

Вибір ігрового рушія. Ознайомившись з ігровими рушіями які є популярними у використанні, мій вибір зупинився на Unity і ось чому:

- безкоштовна версія;
- широкий набір інструментів для роботи з 3D графікою;
- магазин з різноманіттям безкоштовних ігрових ресурсів;
- мова програмування C#.

Вигляд камери.Від першої особи. Ігри з перспективою від першої особи, як правило, засновані на аватарах, де гра відображає те, що аватар гравця бачить на власні очі. Таким чином, гравці зазвичай не можуть бачити тіло аватара, хоча вони можуть бачити зброю або руки аватара. Ця точка зору також часто використовується для представлення точки зору водія в транспортному засобі, як у симуляторах польотів і перегонів; і зазвичай використовують позиційне аудіо, де гучність навколишніх звуків змінюється в залежності від їх положення щодо аватара гравця[17].

Ігри з перспективою від першої особи не вимагають складної анімації для аватара гравця, а також не потребують реалізації ручної або автоматизованої схеми керування камерою, як у перспективі від третьої особи. Перспектива від першої особи дозволяє легше прицілюватися, оскільки немає зображення аватара, яке закривало б погляд гравця. Однак відсутність аватара може ускладнити сприйняття часу та дистанцій, необхідних для стрибків між платформами, і може викликати заколисування у деяких гравців[17].

Гравці очікували, що ігри від першої особи точно масштабують об’єкти до відповідних розмірів. Однак такі ключові об’єкти, як випущені предмети або важелі, можуть бути перебільшені, щоб покращити їх видимість[17].

Стиль графіки.Низькополігональна графіка – це полігональна сітка в комп’ютерній 3D-графіці, яка має відносно невелику кількість полігонів.

Хоча він отримав певну популярність у світі мистецтва, низькополігональна графіка є набагато більш усталеним стилем у відеоіграх, оскільки він служить подвійній меті: скорочення часу розробки та надання грі унікального естетичного стилю.

Сетинг. Фентезі–піджанр фантастики –одного з жанрів сучасного мистецтва, дія якого відбувається у вигаданому світі, де чудеса і вигадка нашого світу є реальністю.

Магія й інші надприродні явища не обов’язково повинні бути головними елементами сюжету, теми чи місця дії, проте в будь-якому випадку вони присутні у творі як частина картини світу. Багато історій цього жанру відбуваються у вигаданих світах, де чари є звичною справою. Як правило, від наукової фантастики та літератури жахів фентезі відрізняється відсутністю (псевдо)-наукових та макабричних тем, хоча усі три жанри мають багато спільного.

У популярній культурі переважає умовно середньовічна форма жанру фентезі, особливо після всесвітнього успіху трилогії «Володар Перснів» Дж. Р. Р. Толкіна. Проте в ширшому, вульгарному розмовному значенні до фентезійного жанру відносять твори багатьох письменників, художників, кінорежисерів і музикантів – починаючи від стародавніх міфів та легенд і закінчуючи сучасними творами, популярними серед широкого загалу.

2.2 Короткий опис сюжету

Головний герой потрапляє до загадкового місця – Безодні – з якого йому потрібно вибратися уникаючи безсмертних ворогів які хочуть його смерті.

Після кожної смерті головного героя світ змінює свою структура, а він повертається до життя, щоб почати спочатку.

2.3 Функціональні схеми

Щоб зобразити можливості гравця при роботі з майбутньою грою була розроблена діаграма прецедентів (рис 2.1)

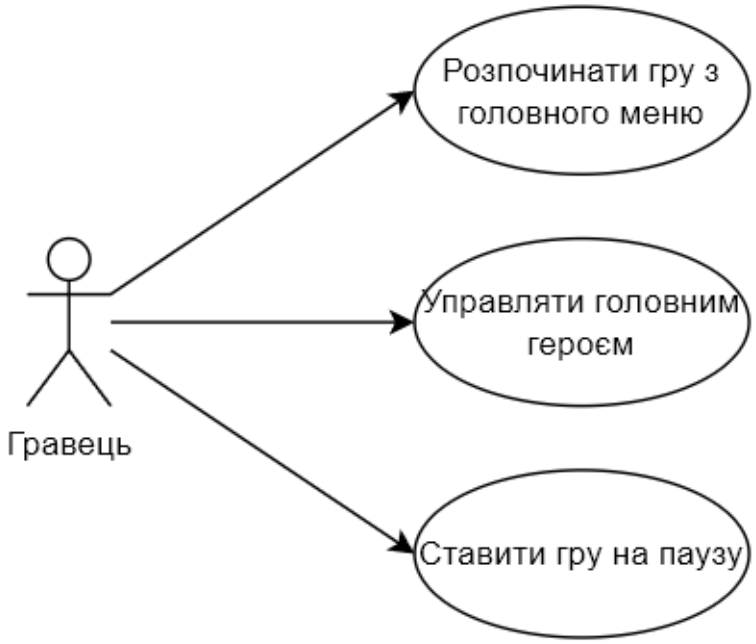

Рисунок 2.1 –Діаграма прецедентів для гравця

Гравець має наступні можливості:

- Розпочинати гру з головного меню.
- Управляти головним героєм.
- Ставити гру на паузу.

Щоб зобразити можливості ворога у майбутній грі була розроблена діаграма прецедентів (рис 2.2)

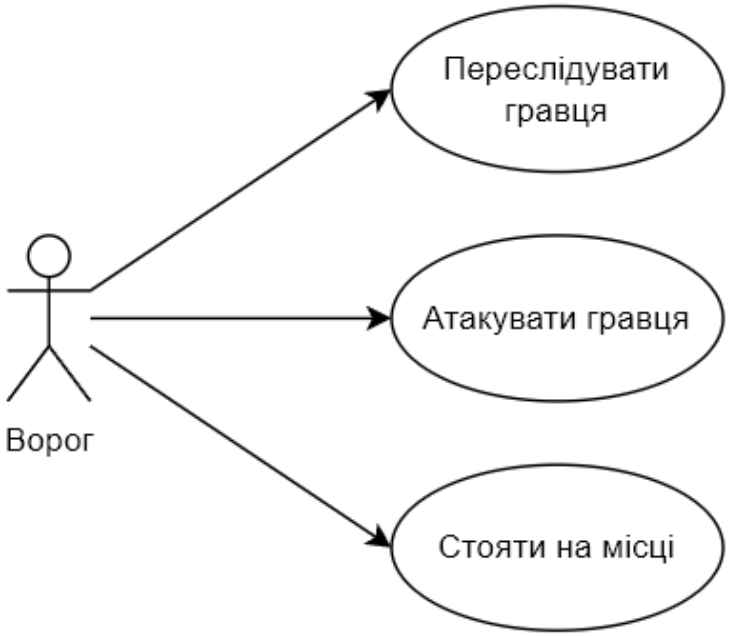

Рисунок 2.2 –Діаграма прецедентів для ворога

Ворог має наступні можливості:

- Переслідувати гравця.
- Атакувати гравця.
- Стояти на місці.

2.3 Проєктування кімнат гри

Рівні гри генеруються з заздалегідь створених кімнат, які об’єднані між собою. Кімнати якими починається та закінчується рівень використовуються 1 раз.

На рисунка 2.3-2.8 представлені схеми кімнат, де:

- чорними лініями зображені границі кімнати (стіни, решітки, колони та інші об’єкти, які можуть виступати в ролі границь);

- зеленими лініями позначені можливі проходи;
- червоні багатокутники – місця можливої появи ворогів.

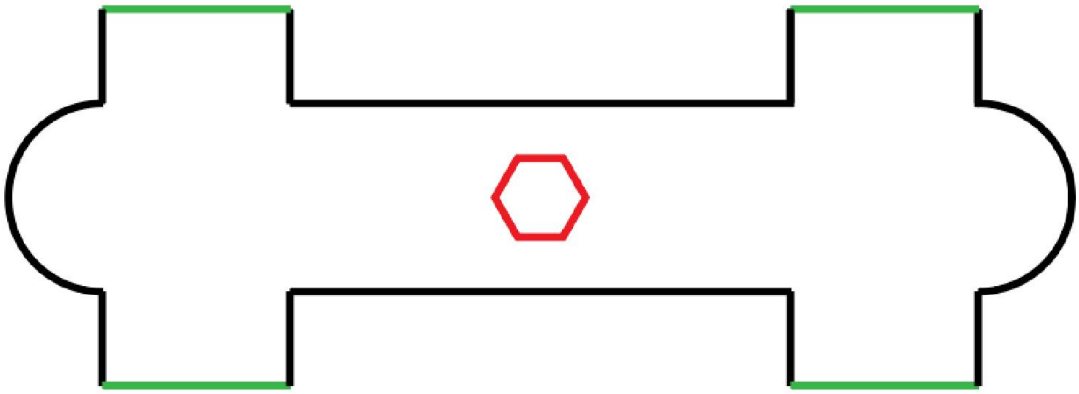

Рисунок 2.3–Схема кімнати№1

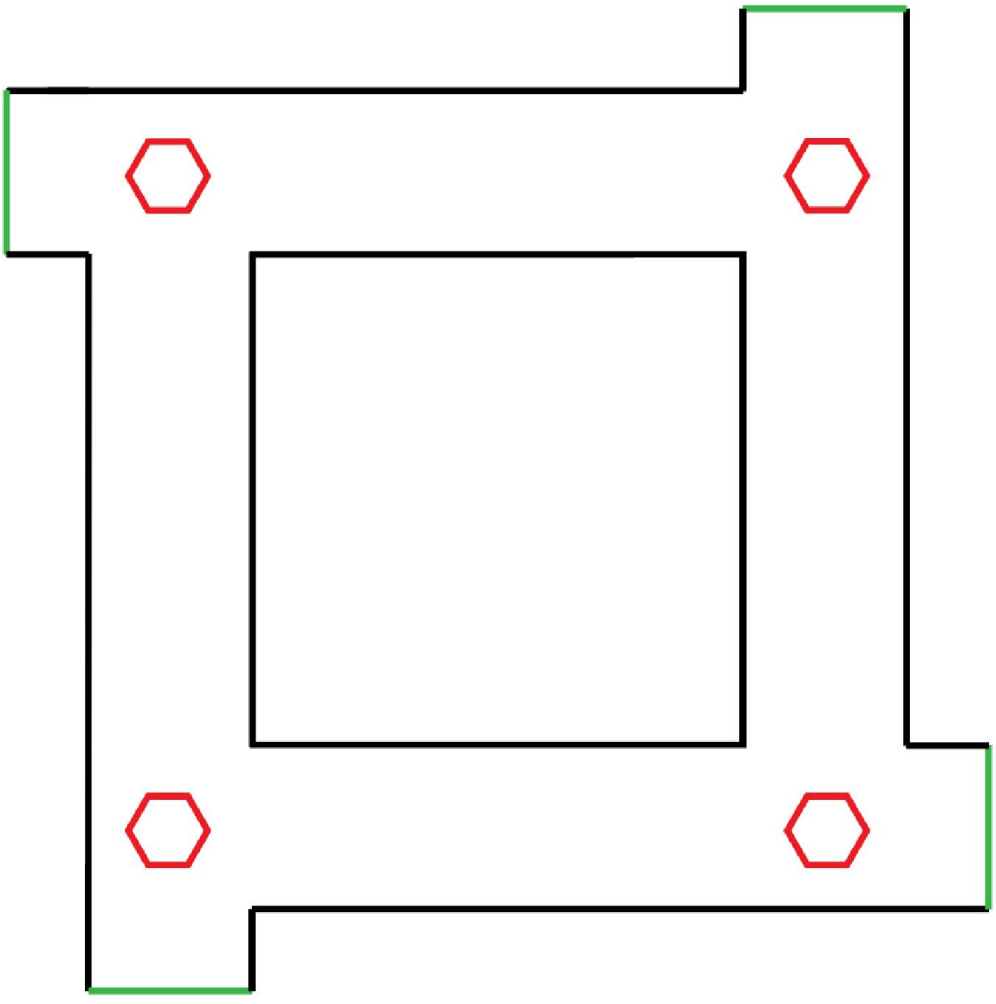

Рисунок 2.4–Схема кімнати №2

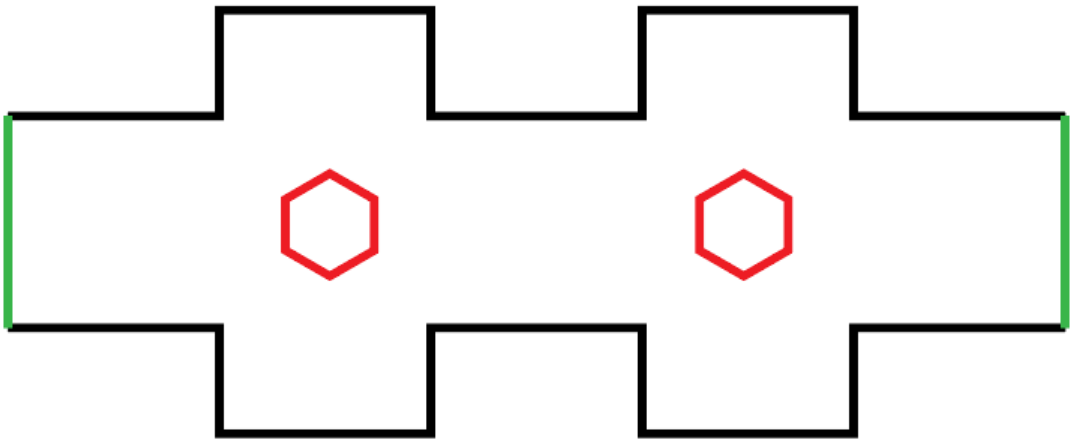

Рисунок 2.5–Схема кімнати №3

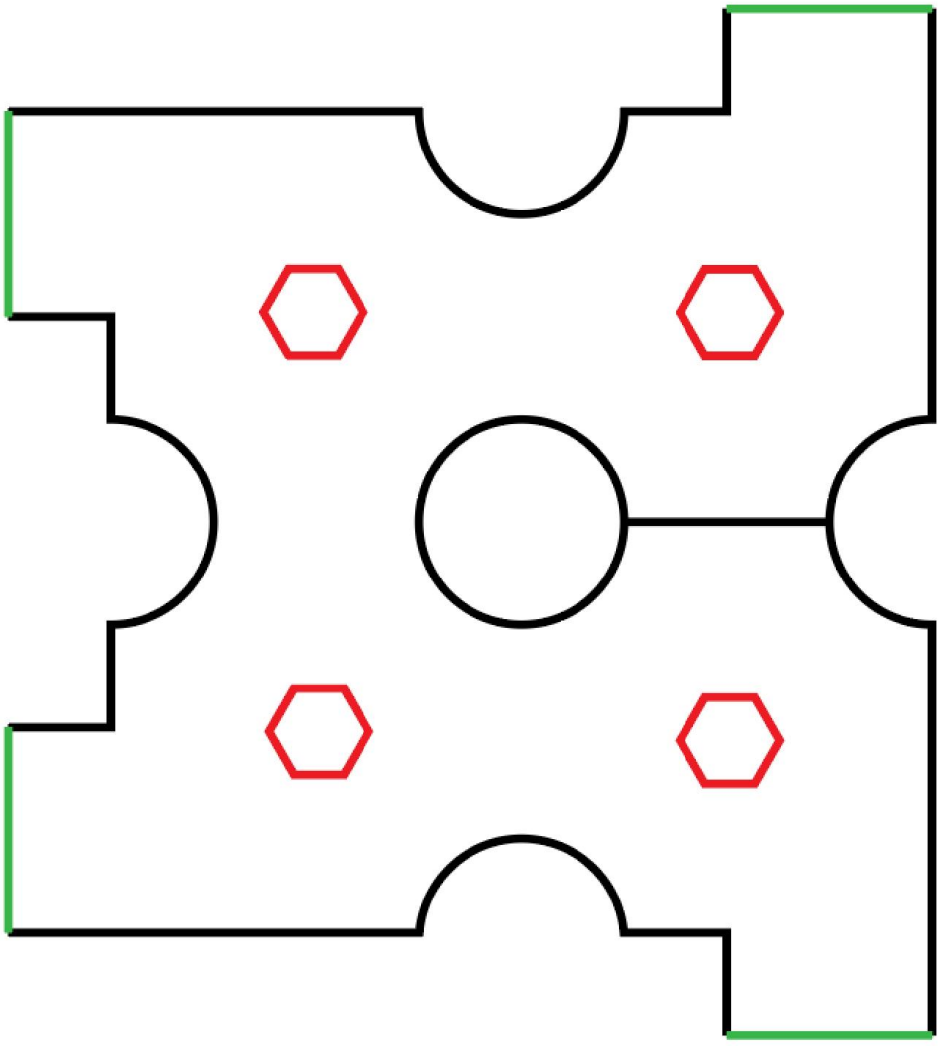

Рисунок 2.6–Схема кімнати №4

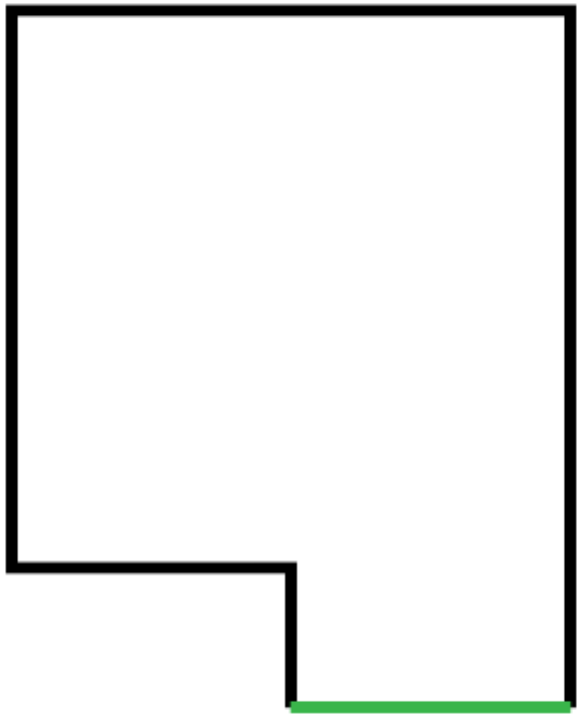

Рисунок 2.7–Схема початкової кімнати

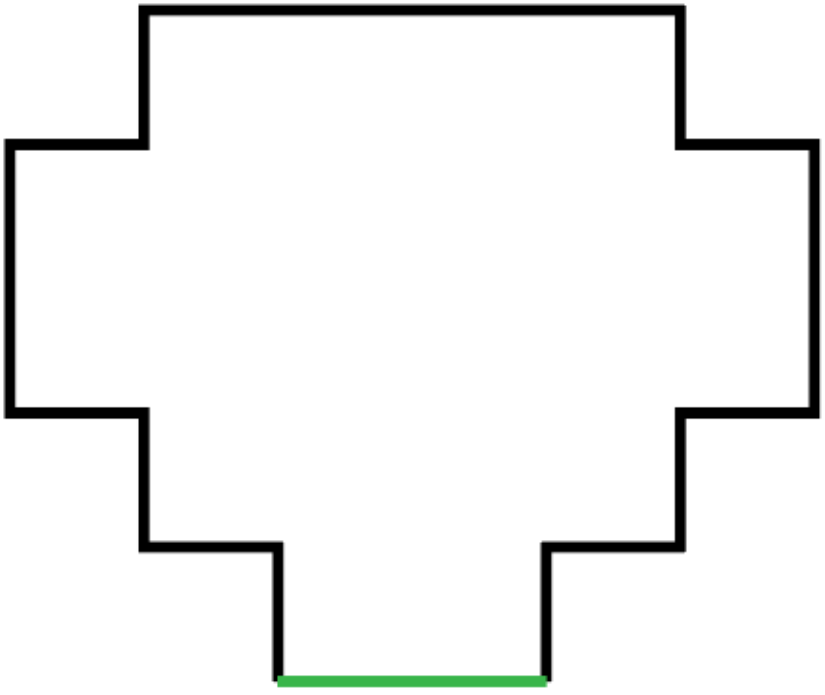

Рисунок 2.8–Схема кінцевої кімнати

2.4 Спосіб генерації рівнів

Для генерації рівнів в іграх жанру roguelike можуть використовуватися безліч різних алгоритмів. Але можна виділити два основні типи генерації: процедурна та з використанням заготовлених кімнат.

Процедурна генерація – автоматичне створення ігрового контенту за допомогою алгоритмів. Іншими словами, процедурна генерація є програмним забезпеченням, яке може створювати ігровий контент самостійно, або спільно при взаємодії з гравцями або геймдизайнерами. Під контентом розуміється створення рівнів гри, карти ігрового світу, правил гри, текстур, сюжетів, предметів, завдань, музики, зброї, транспортних засобів, персонажів та інше[18].

Як характерні приклади використання процедурної генерації називаються: генерація без участі людини підземель у пригодницьких ігор, що отримує новий світ на кожному запуску; генерація повної, грабельної та збалансованої настільної гри; внутрішня процедура ігрового движка, що швидко заповнює ігровий світ рослинами; інструмент, що дає можливість створювати людині карти для стратегічної гри, і при запитах і заданих змінах, що перераховує карту для її поліпшення, а також пропонує варіанти, що дозволяють зробити карту більш збалансованою та цікавою[18].

Метою використання процедурної генерації може бути створення ігрового контенту без участі людини розробка інших типів ігор, адаптація ігор під гравця «на льоту», покращення контенту за допомогою алгоритмічних рішень, а також формалізація геймдизайну як широке наукове завдання .

Використання раніше заготовлених кімнат дає смогу більш влучно передати атмосферу того, що відбувається на екрані; виключає появу значної кількості помилок у генерації. Схематичний приклад на рис. 2.9.

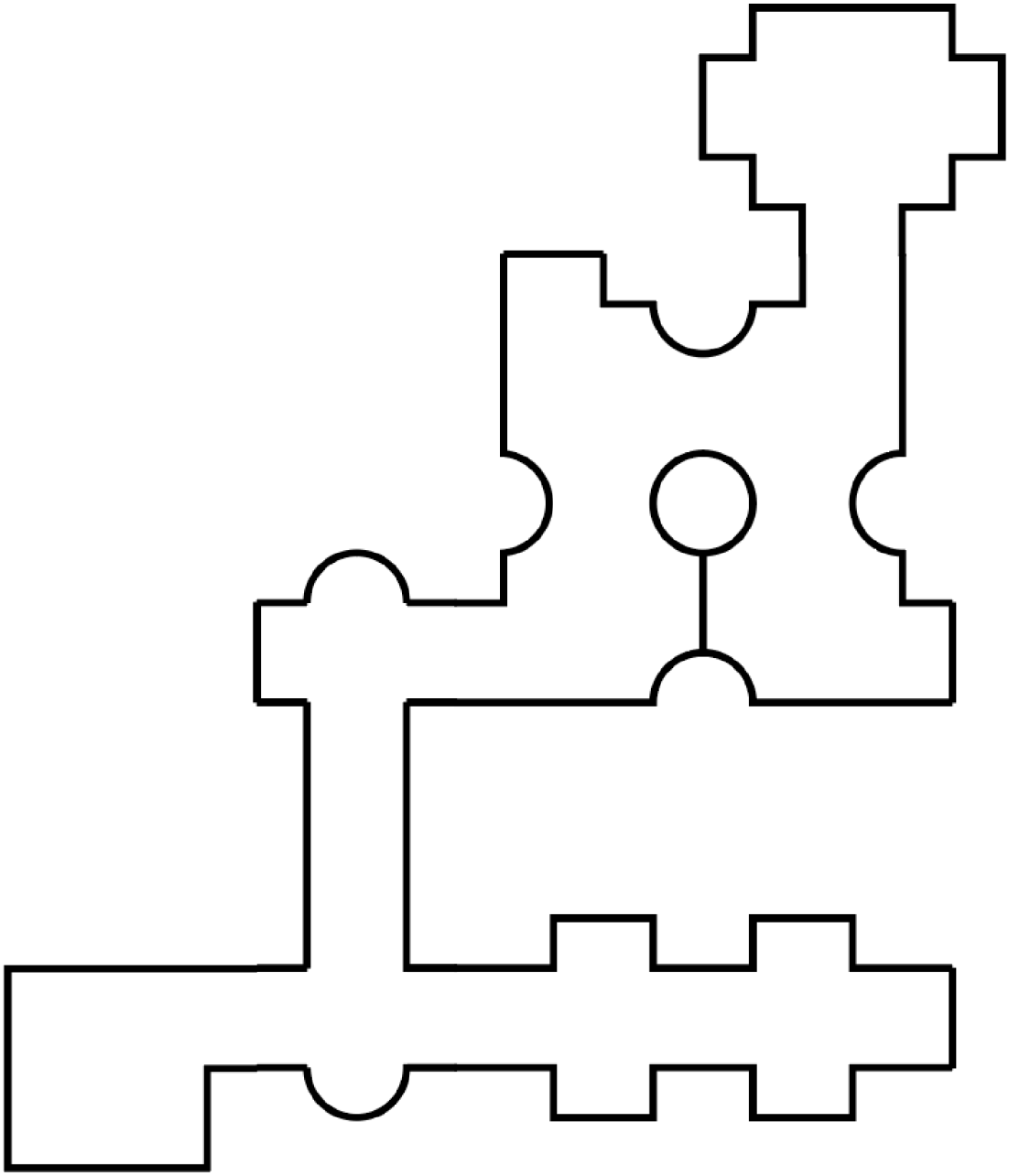

Рисунок 2.9–Схематична робота генератора рівнів

Логіку роботи генератора рівнів можна відобразити блок-схемою, яка зображена на рис. 2.10.

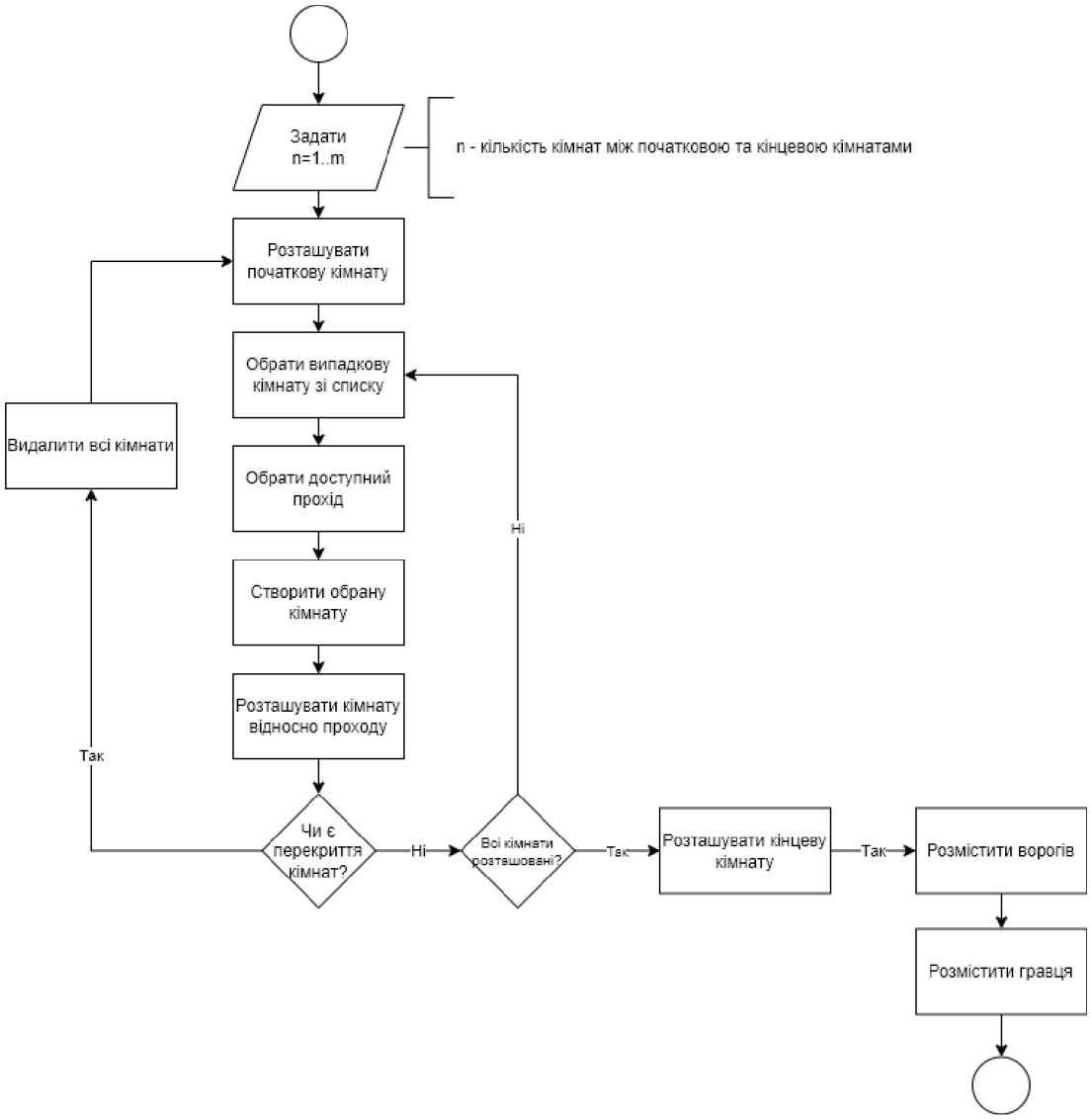

Рисунок 2.10–Блок-схема роботи генератора рівнів

2.5 Проєктування роботи штучного інтелекту ворогів

Ігровий штучний інтелект –набір програмних методик, які використовуються у відеоіграх для створення ілюзії інтелекту в поведінці персонажів, керованих комп’ютером. Ігровий ШІ, крім методів традиційного штучного інтелекту, включає також алгоритми теорії керування, робототехніки, комп’ютерної графіки та інформатики у цілому[19].

Реалізація ШІ сильно впливає на ігровий процес, системні вимоги і бюджет гри, і розробники балансують між цими вимогами, намагаючись зробити цікавий і невимогливий до ресурсів ШІ малою ціною. Тому підхід до ігрового ШІ серйозно відрізняється від підходу до традиційного ШІ –широко застосовуються різного роду спрощення, обмани й емуляції. Наприклад: з одного боку, в шутерах від першої особи безпомилковий рух і миттєве прицілювання, властиве ботам, не залишає жодного шансу людині, так що ці здатності штучно знижуються. З іншого боку –боти повинні робити засідки, діяти командою й т.д., для цього застосовуються «костилі» у вигляді контрольних точок, розставлених на рівні[19].

Персонажів відеоігор, керованих ігровим штучним інтелектом, ділять на:

1. Неігрові персонажі –зазвичай, ці ШІ-персонажі є дружніми або нейтральними до людського гравця;

2. Боти –ворожі до гравця ШІ-персонажі, що наближаються за можливостями до ігрового персонажа; проти гравця в будь-який конкретний момент бореться невелика кількість ботів. Боти найскладніші в програмуванні.

3. Моби–ворожі до гравця «низькоінтелектуальні» ШІ-персонажі. Моби вбиваються гравцями у великих кількостях заради очок досвіду, артефактів або проходження території.

Логіку роботи штучного інтелекту ворогів можна відобразити блок-схемою, яка зображена на рис. 2.11.

Рисунок 2.11–Блок-схема роботи штучного інтелекту ворогів

Роботу штучного інтелекту ворогів в майбутній грі схематично можна зобразити рисунком 2.12, де:

- чорний многокутник є ворогом;
- червоне коло – зона в якій ворог починає атакувати гравця;
- сине коло – зона в якій ворог починає переслідування гравця.

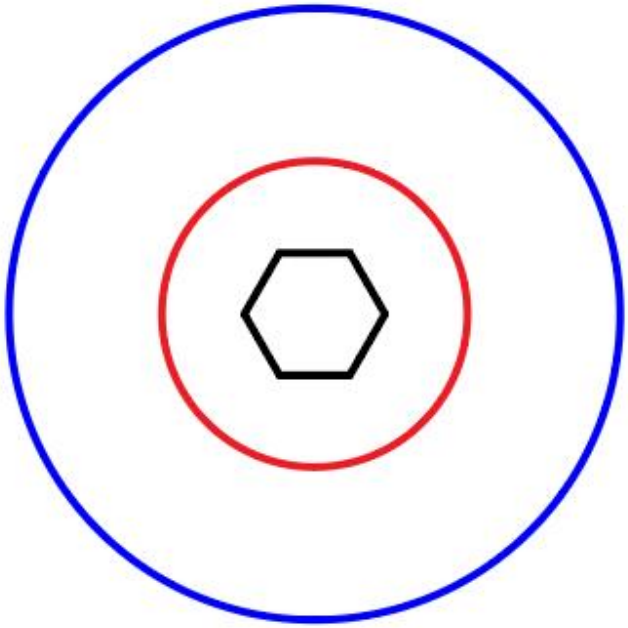

Рисунок 2.12–Схема логіки ворога

2.6 Додаткові проєктні рішення

Пошук шляху (англ. Pathfinding) –термін в інформатиці та штучному інтелекті, що означає визначення комп'ютерною програмою найкращого, оптимального маршруту між двома точками[20].

Пошук шляху в контексті комп'ютерних ігор стосується шляху, на якому об'єкт, що рухається, шукає шлях навколо перешкод. Найчастіше завдання пошуку шляху виникає у стратегіях реального часу, у яких гравець дає завдання ігровим юнітам (одиницям) рухатися через ігровий рівень, що містить перешкоди. Крім стратегій, завдання пошуку шляху так чи інакше тією чи

іншою мірою зустрічається в більшості сучасних ігрових жанрів. Так як ігри стають все складнішими, то пошук шляху також еволюціонує та розвивається разом з ними[20].

Стратегії реального часу зазвичай містять великі території з відкритим ландшафтом, у яких пошук шляху зазвичай є простим завданням. Однак у більшості випадків картою переміщається не один юніт, а кілька, що створює потребу в різних і набагато складніших алгоритмах пошуку шляху для уникнення пробок у вузьких областях ігрового ландшафту. У стратегіях ігровий рівень ділиться на тайли, які діють як вузли алгоритмі пошуку шляху[20].

У жанрі 3D-шутерів використовуються набагато обмеженіші простори, які не так легко розділити на вузли. Тут замість вузлів використовуються звані точки шляху. Точки шляху –це нерегулярні та вручну встановлені вузли, які містять інформацію про те, до яких інших вузлів можна дістатися[20].

Алгоритм пошуку A* –в інформатиці та математиці, алгоритм пошуку за першим найкращим збігом на графі, який знаходить маршрут з найменшою вартістю від однієї вершини (початкової) до іншої (цільовий, кінцевий)[21].

Приклад роботи алгоритму на рис. 2.13, де:

- зеленим позначена початкова точка;
- червоним – шлях;
- синім – кінцева точка;
- сірим – перешкода.

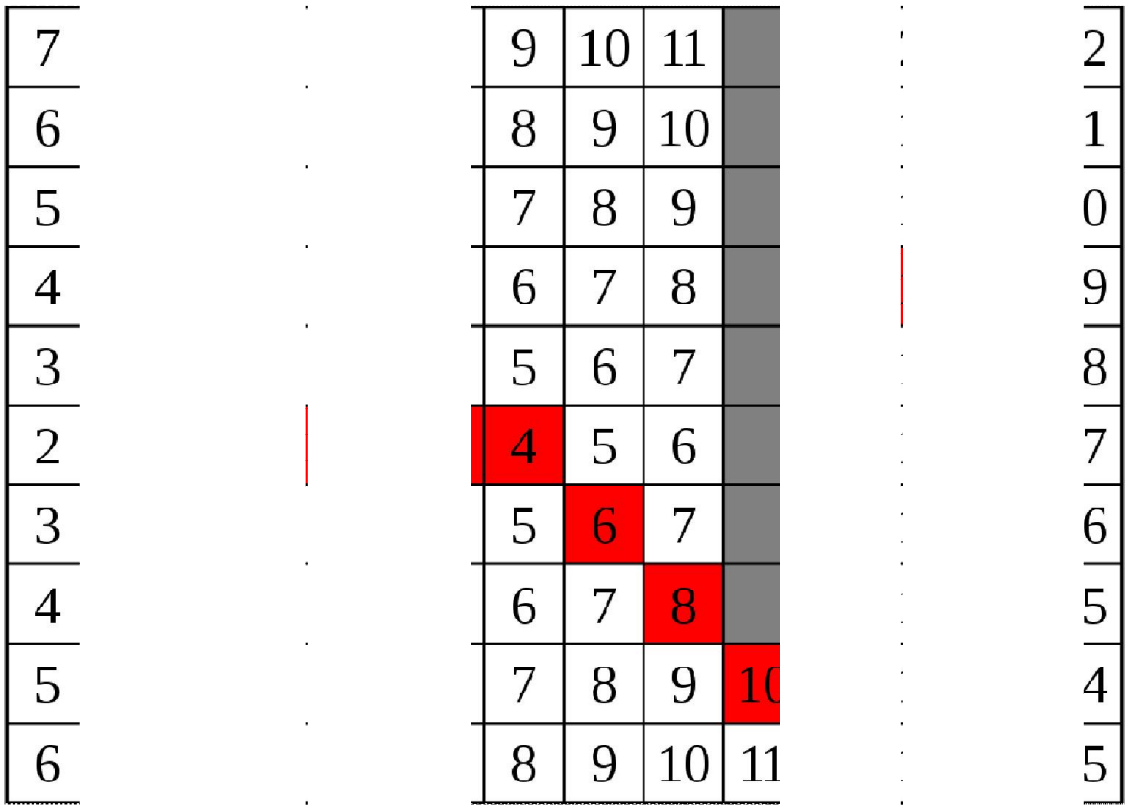

Рисунок 2.13–Приклад роботи алгоритму пошуку A*

Порядок обходу вершин визначається евристичною функцією «відстань + вартість» (як правило, що позначається як f(x)). Ця функція –сума двох інших: функції вартості досягнення вершини (x) з початкової (зазвичай позначається як g(x) і може бути як евристичної, так і ні), і функції евристичної оцінки відстані від розглянутої вершини до кінцевої (позначається як h (x))[21].

Функція h(x) повинна бути припустимою евристичною оцінкою, тобто не повинна переоцінювати відстань до цільової вершини. Наприклад, для задачі маршрутизації h(x) може бути відстань до мети по прямій лінії, так як це фізично найменша можлива відстань між двома точками[21].

Алгоритм A* спершу відвідує ті вершини, які ймовірно ведуть до найкоротшого шляху до мети. Аби розпізнати такі вершини, кожній відомій вершині x співставляється значення f(x), яке дорівнює довжині найкоротшого

шляху від початкової вершини до кінцевої, який пролягає через обрану вершину. Вершини з найменшим значенням f обираються в першу чергу[21].

Функція f(x) для вершини x визначається f(x)=g(x)+h(x), де:

- g(x) функція, значення якої дорівнюють вартості шляху від початкової вершини до x,
- h(x) евристична функція, оцінює вартість шляху від вершини x до кінцевої.

Використана евристика не повинна давати завищену оцінку вартості шляху. Прикладом оцінки може служити пряма лінія: загальний шлях не може бути коротшим за пряму лінію.

Алгоритм ділить вершини на три класи:

- невідомі вершини: ці вершини ще не були знайдені. Ще не відомий шлях до них. На початку роботи алгоритму всі вершини, окрім початкової, належать до класу невідомих;
- відомі вершини (OpenList): вже відомий (можливо не оптимальний) шлях до цих вершин. Всі відомі вершини разом зі значеннями f зберігаються в списку. З цього списку вибираються, в першу чергу, найперспективніші вершини. Конкретна реалізація цього списку має суттєвий вплив на швидкодію алгоритму, і зазвичай має вигляд черги з пріоритетом (наприклад, бінарна купа). На початку роботи алгоритму до відомих вершин належить лише початкова вершина;
- повністю досліджені вершини (ClosedList): до цих вершин вже відомий найкоротший шлях. Повністю досліджені вершини додаються до так званого замкненого списку, аби запобігти багаторазовому дослідженню вже досліджених вершин. Список повністю досліджених вершин на початку роботи алгоритму порожній.

Кожна відома або повністю досліджена вершина має вказівник на попередні вершини. Завдяки цьому вказівникові, можна пройти шляхом від цієї до початкової вершини.

Коли вершину x буде повністю досліджено, суміжні з нею вершини додаються до списку відомих вершин, а сама вершина додається в список повністю досліджених. Вказівники на попередню вершину встановлюються на x. Суміжні вершини, які вже знаходяться в списку повністю досліджених вершин, до списку відомих не додаються, а зворотні вказівники не змінюються. Суміжні вершини, які вже знаходяться в списку відомих, лише оновлюються (значення f та вказівник на попередню вершину), якщо знайдений до них шлях коротший за вже відомий.

Алгоритм зупиняється коли кінцева вершина потрапляє до списку повністю досліджених вершин. Знайдений шлях відтворюється за допомогою вказівників на попередню вершину. Якщо список відомих вершин порожніє, то розв’язку задачі не існує і алгоритм припиняє пошук.

Відтворений за зворотніми вказівниками знайдений шлях починається з кінцевої вершини та прямує до початкової. Аби одразу отримати шлях в правильному напрямі, з початкової вершини до кінцевої, в умовах задачі слід переставити місцями початок та кінець. Якщо шукати шлях починаючи з кінцевої вершини, відтворений список починатиметься з початкової вершини й прямуватиме до кінцевої[21].

2.7 Вимоги до відеогри

Відеогра повинна реалізовувати наступне:

- Два режими «Меню» та «Ігровий».
- В режимі «Меню» повинні бути кнопки «Вихід» та «Розпочати гру».

- «Ігровий» режим активується після натискання кнопки «Розпочати гру».
- В «Ігровому» режимі відбувається наступне:
  - Генерується рівень з заздалегідь визначених кімната зі списку таких кімнат.
  - З’являються вороги у кімнатах.
  - З’являється гравець.
  - Щоб закінчити рівень потрібно дістатися до кінця рівня.
- В «Ігровому» режимі повинен бути інтерфейс користувача який показує здоров’я героя.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1 Візуальна складова

Моделі ігрових об’єктів та анімації були взяті з магазину Unity3DAssetStore. А саме збірки: «POLYGON – DungeonsPack v1.3» (ігрові об’єкти), «RPG CharacterMecanimAnimationPack v6.0» (анімації), плагін «A*Pathfinding Project Pro v4.2.10» (пошук шляху A*). На основі цих збірок були створені неігрові персонажі (рис. 3.1-3.2) та кімнати (рис. 3.3-3.8).

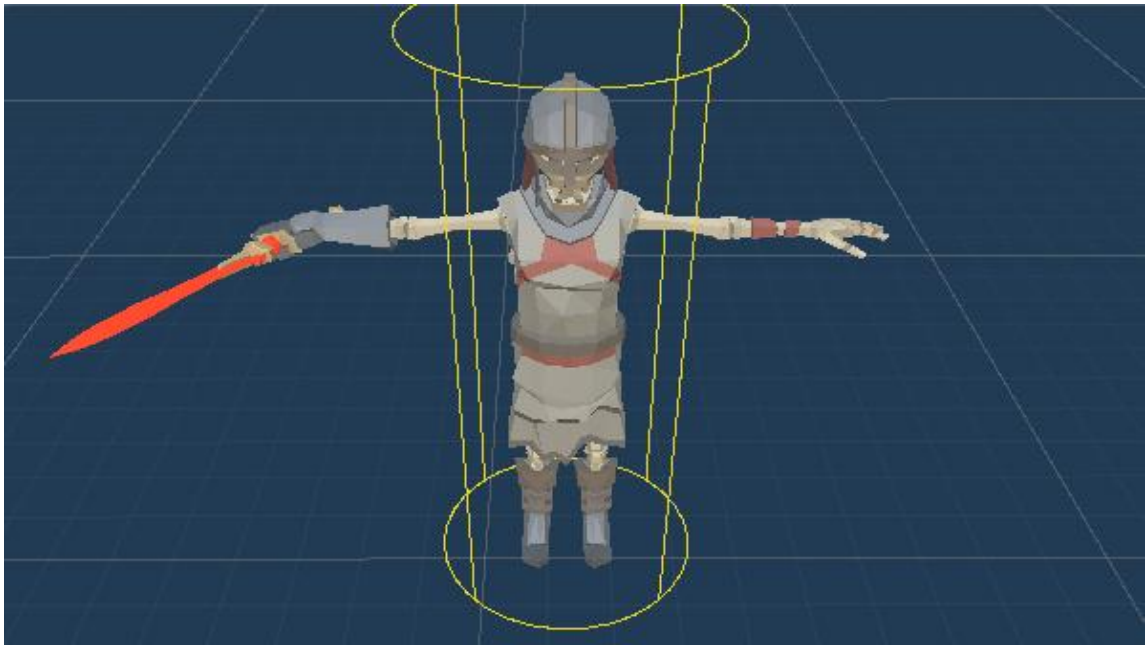

Рисунок 3.1 – Модель скелету

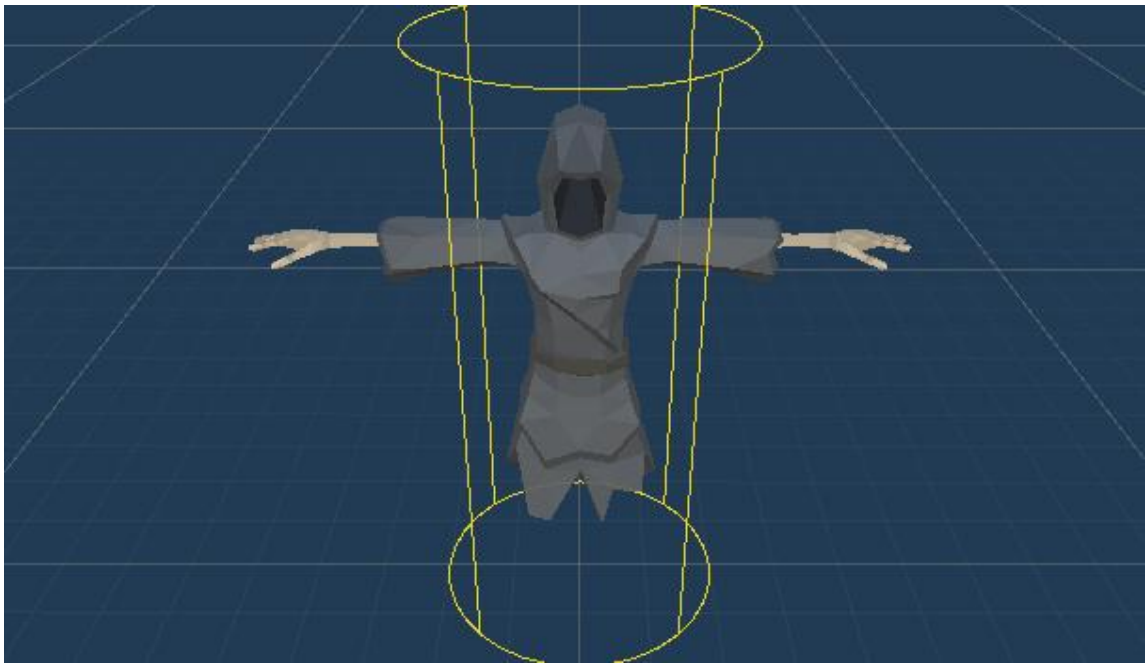

Рисунок 3.2 – Модель душі

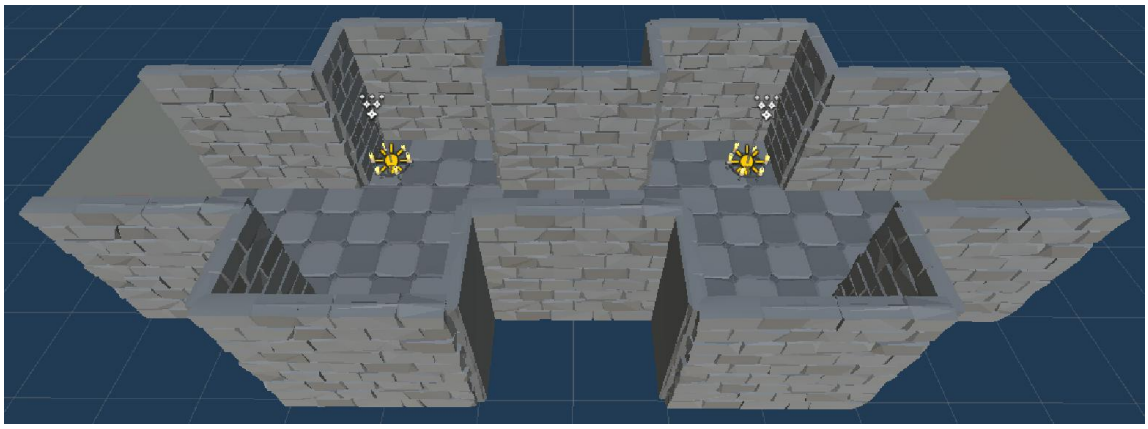

Рисунок 3.3 – Модель кімнати №1

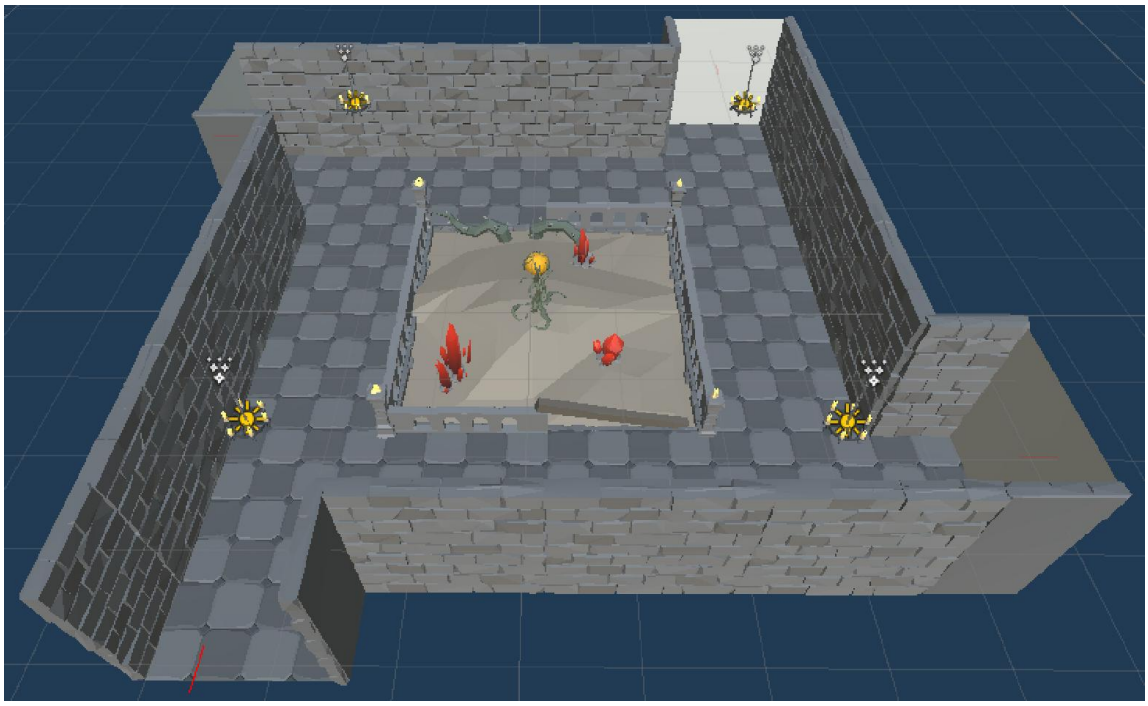

Рисунок 3.4 – Модель кімнати №2

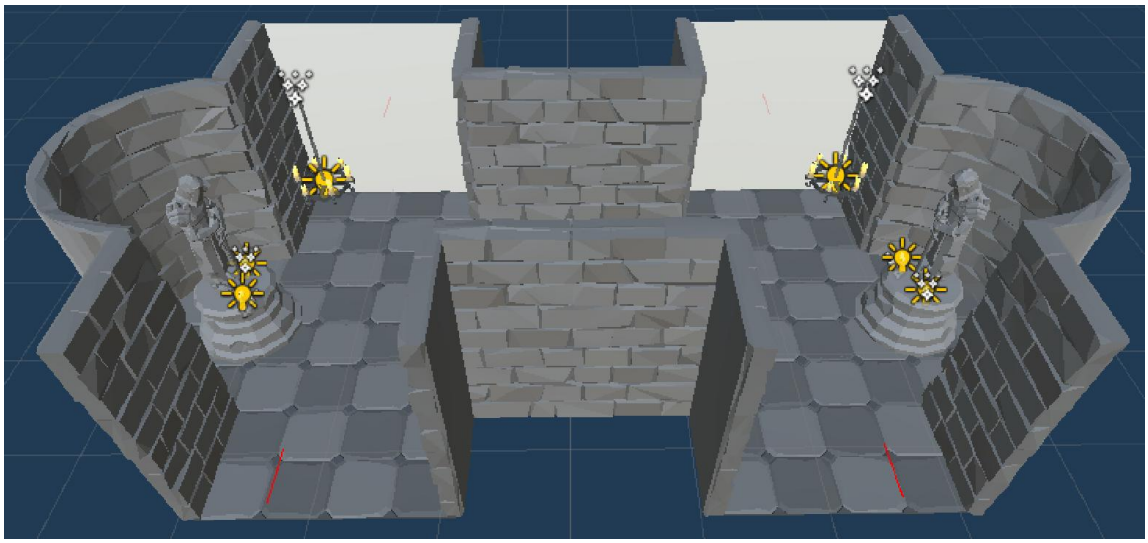

Рисунок 3.5 – Модель кімнати №3

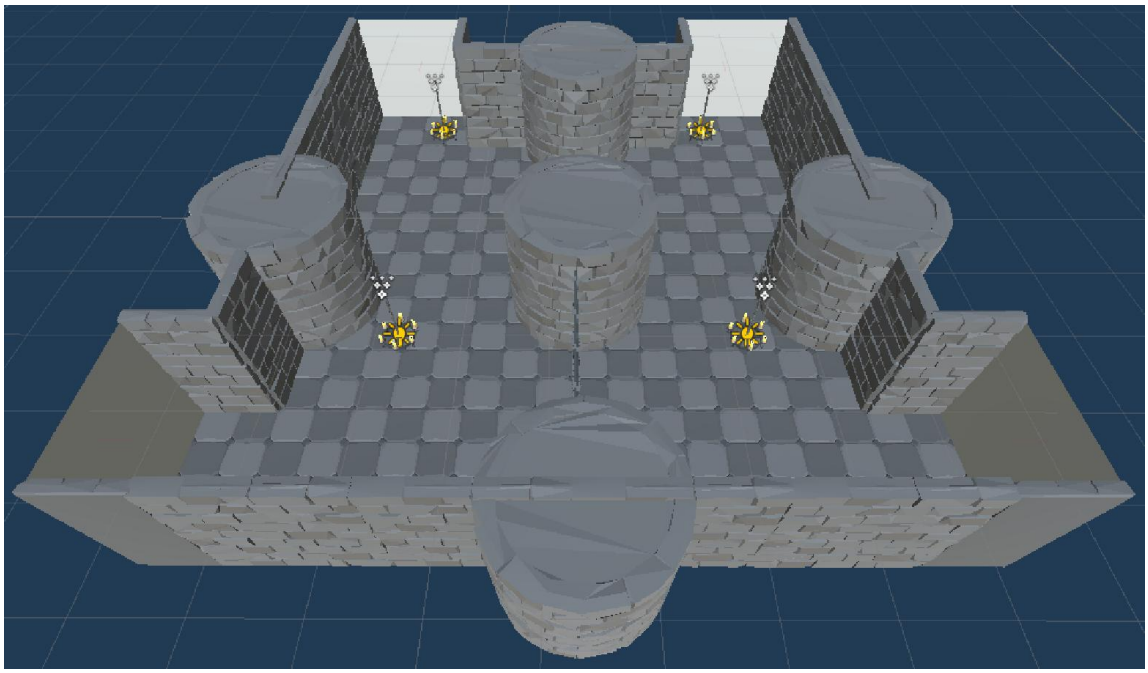

Рисунок 3.6 – Модель кімнати №4

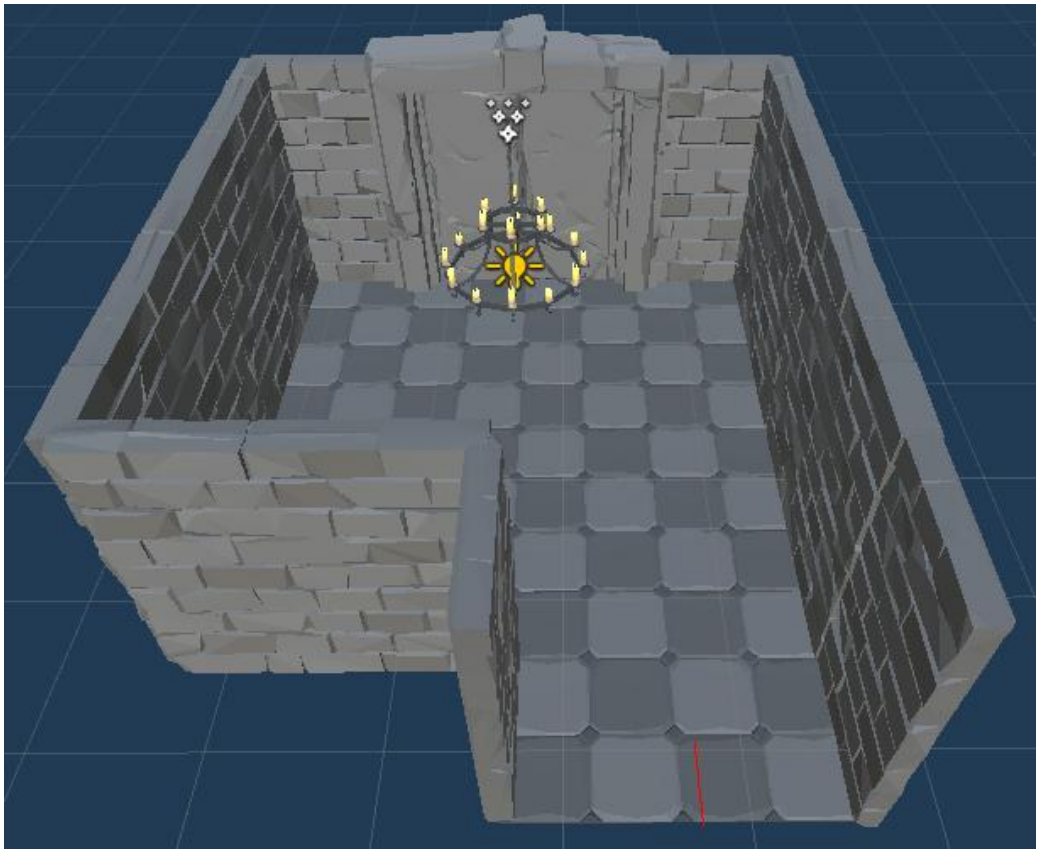

Рисунок 3.7 – Модель початкової кімнати

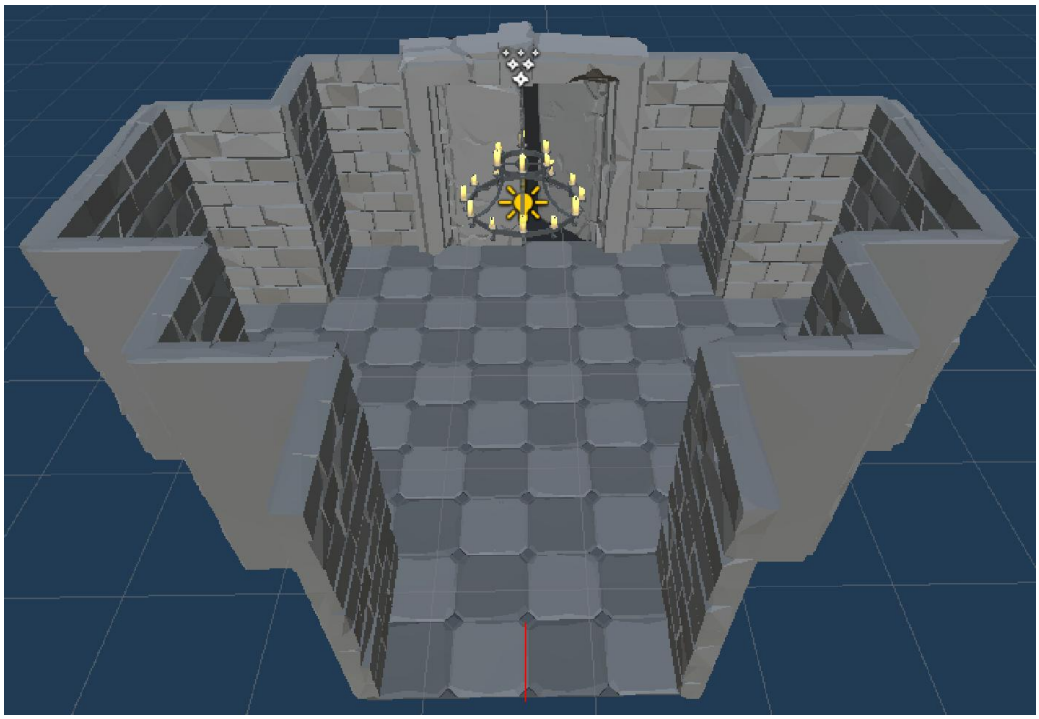

Рисунок 3.8 – Модель кінцевої кімнати

3.2 Генератор рівнів

Частинаскриптугенератора (Повний скрипт у Додатоку А):

```
…
IEnumeratorGenerateLevel()
   {
WaitForSecondsstartup= new WaitForSeconds(1);
WaitForFixedUpdate interval = new WaitForFixedUpdate();
yield return startup;

PlaceStartRoom();//Розташуватипочатковукімнату
yield return interval;
int iterations = Random.Range((int) iterationRange.x, (int) iterationRange.y);
//Обрати випадкову кімнату зі списку
```

```
for (inti = 0; i< iterations; i++)
        {
PlaceRoom(); //Розташувати обрану кімнату
yield return interval;
        }
PlaceEndRoom(); //Розташувати кінцеву кімнату
yield return interval;
    }
…
```

Приклади генерації на рис. 3.19-3.11.

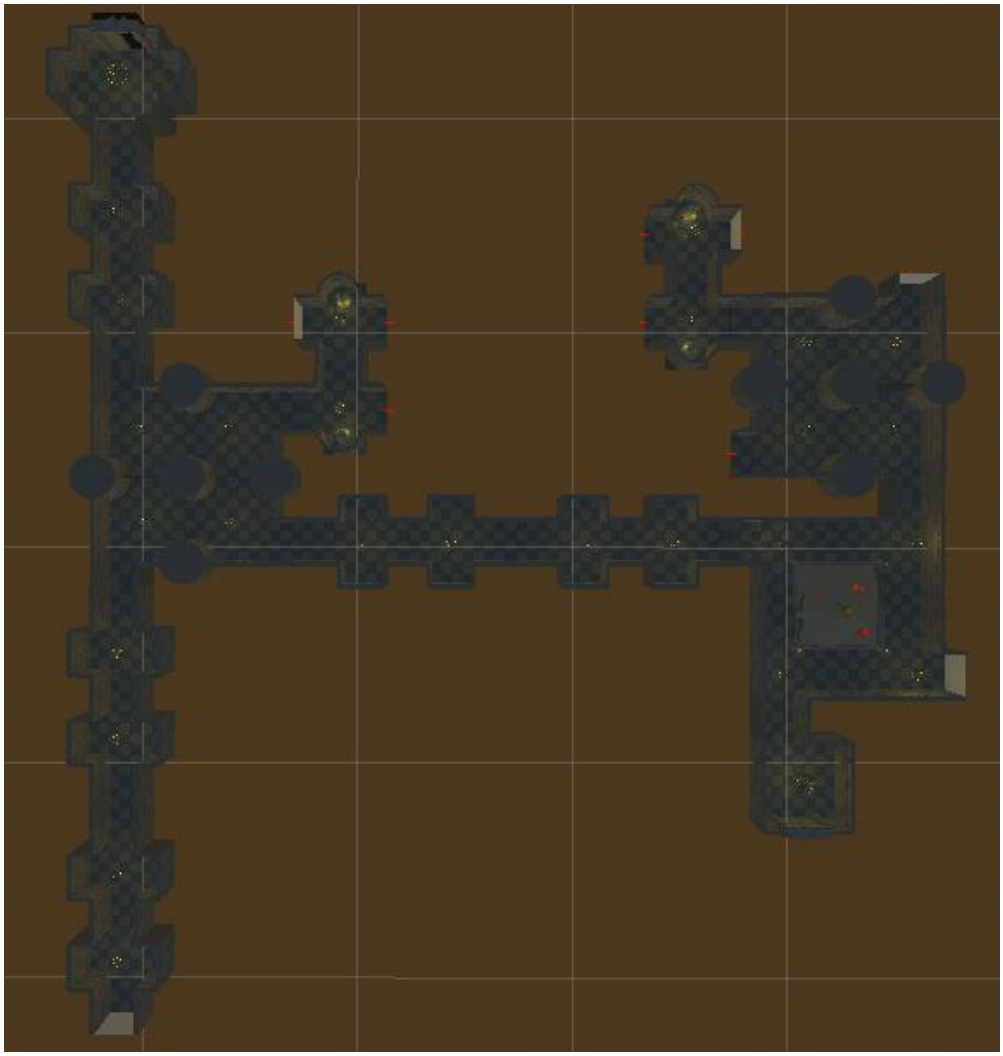

Рисунок 3.9 – Приклад генерації №1

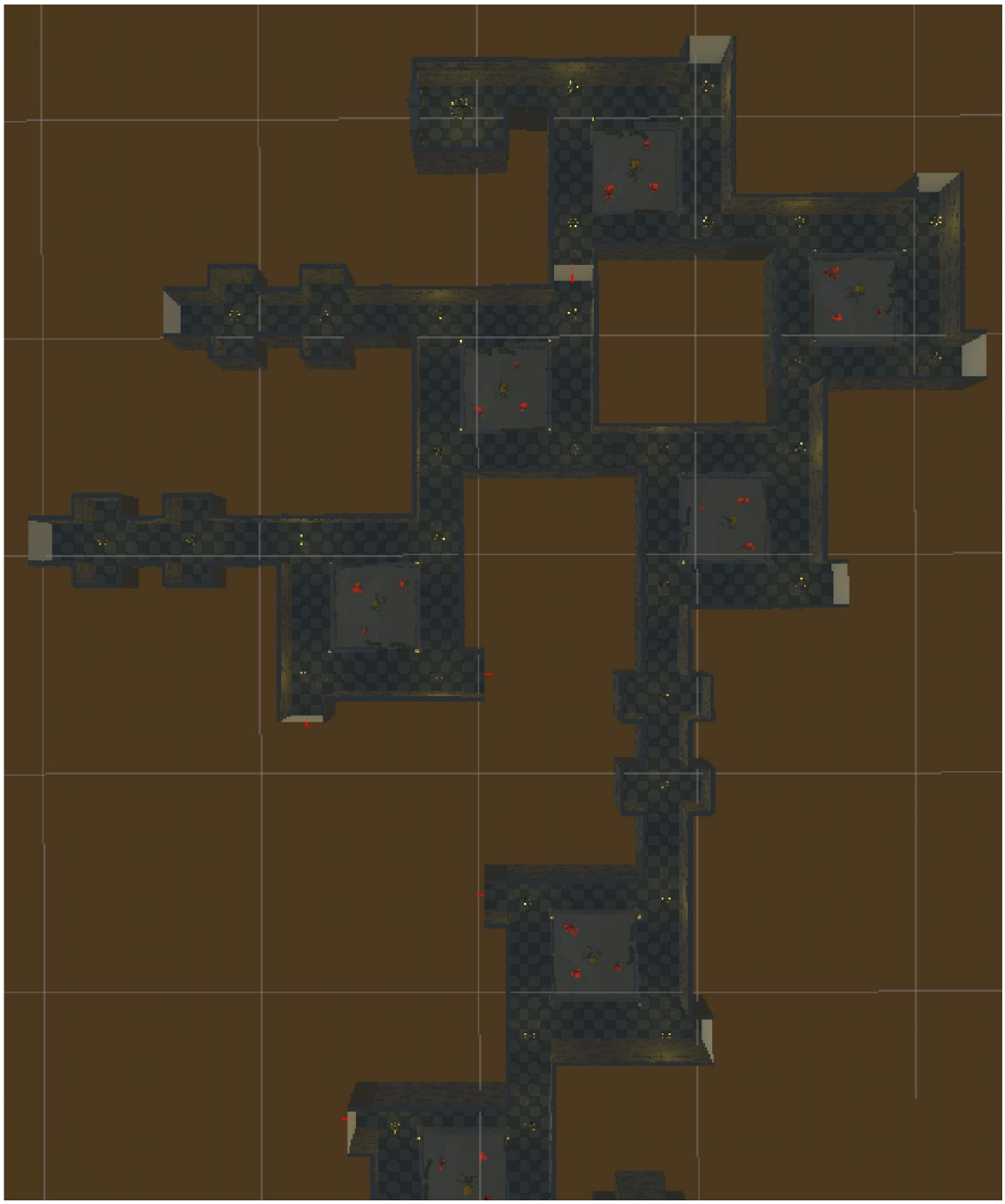

Рисунок 3.10 – Приклад генерації №2

Рисунок 3.11 – Приклад генерації №3

3.3 Робота штучного інтелекту ворогів

Вороги мають спільний код, що відповідає за їх поведінку.

Частинаскрипту управління ворогомна прикладі скелету(Повний скрипт у Додатоку А):

…

privatevoidLogicUpdate()

```
{
if (DeathInfo) //Якщо мертвий, то нічого не робити
return;
if (AttackInfo) //Поки ворог атакує, він не може робити інші дії
{
    _swordCollider.enabled = true;
return;
  }
HandleAnimations(); //Обробник анімацій
  _swordCollider.enabled = false;
playerInSightRange = CollisionDetection(sightRange, playerLayer); //Чи знаходить гравець в зоні видимості
playerInAttackRange = CollisionDetection(attackRange, playerLayer); //Чи знаходиться гравець в зоні атаки
if (playerInSightRange&&playerInAttackRange) //Умови для атаки гравця
  {
MoveTo(transform.position); //Стояти на місці
Attack(player); //Атакувати гравця
  }
if(!playerInSightRange&& !playerInAttackRange) //Умови для бездіяльності
{
MoveTo(transform.position); //Стояти на місці
  }
if (playerInSightRange&& !playerInAttackRange) //Умови для переслідування гравця
  {
ChaseTarget(player); //Переслідувати гравця
  }
```

}

…

На рис. 3.12 зображений внутрішньоігровий вигляд роботи скрипту вище, де жовтою сферою позначено зона видимості, а червоною – зона атаки.

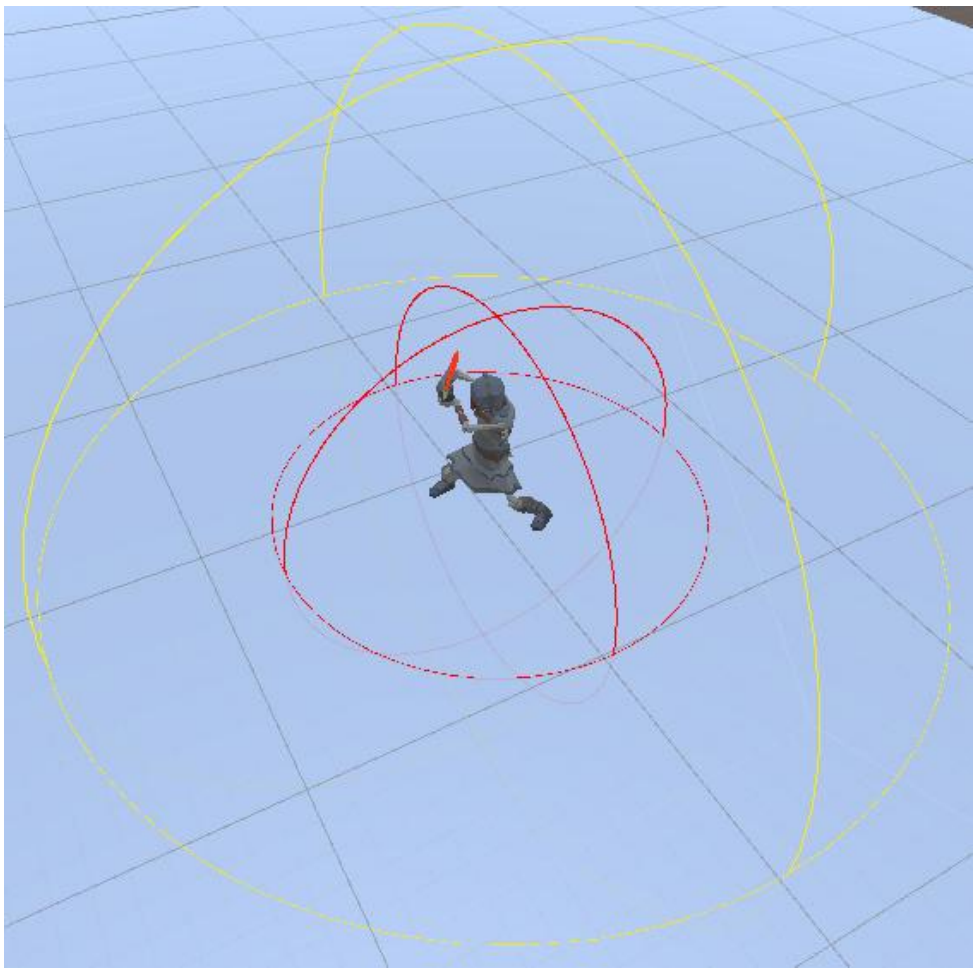

Рисунок 3.12 – Робота скрипту контролю ворога у режимі «Розробника»

3.4 Створення ігрового процесу

Головне меню. Запустивши гру, гравець потрапляє до головного меню(рис 3.13). Після натискання кнопки «Розпочати гру», відбувається генерація рівня та розміщення головного героя на ньому. Натиснувши кнопку «Вихід», гравець закриває гру та повертається до робочого столу.

Рисунок 3.13 – Головне меню гри

Пересування головного героя. Персонаж може здійснювати рух у будь-якому напрямку в тривимірному просторі. Тому гравець не може проходити крізь стіни тому що, виконується взаємодія фізичних об’єктів. Швидкість руху персонажа статична та відповідає нешвидкій пробіжці.

Бойова система.Головнийгерой не має зброї та не може наносити шкоди ворогам чи оточенню. Він може лише отримувати пошкодження, яке зменшує кількість життів. Вороги в свою чергу можуть атакувати героя.

Інтерфейс. На екрані користувача знаходиться лише здоров’я головного героя.

Взаємодія із ігровими об’єктами. Із взаємодій можна лише відзначити тригер закінчення рівня у кінцевій кімнаті та невидимі стіні, що гравець не міг зайти куди не потрібно (рис. 3.14-3.15).

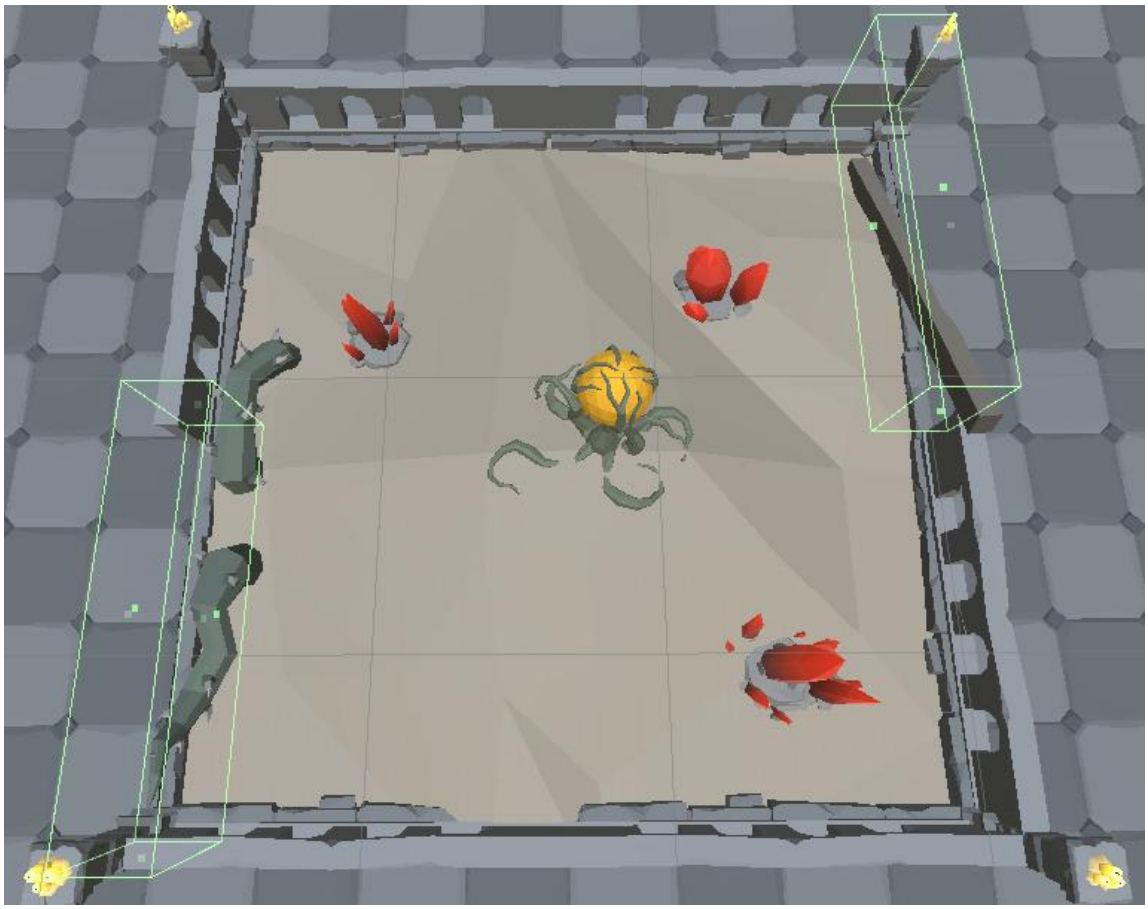

Рисунок 3.14 – Невидимі стіни

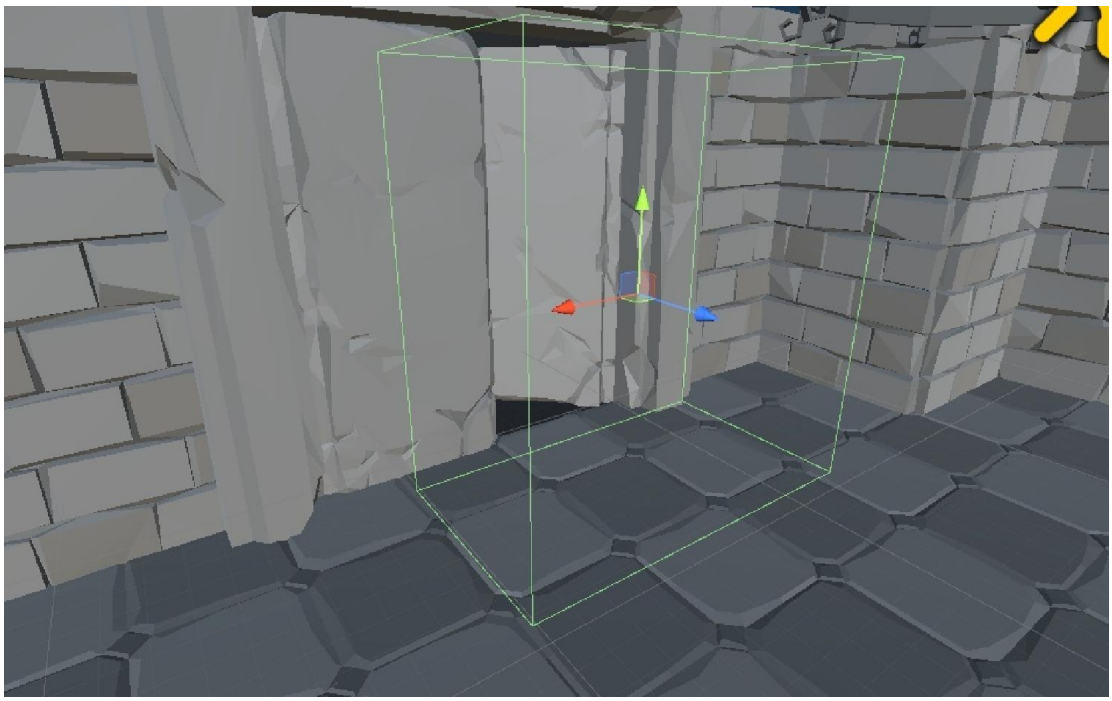

Рисунок 3.15 – Тригер кінця рівня

Джерела світла. Так як у грі багато об’єктів освітлення використовують свічки, тому світло має жовтий колір та не є дуже яскравим (рис. 3.16).

Рисунок 3.16 – Джерела світла

Звуковий супровід. Усі основні ігрові об’єкти мають свої звуки. У грі є фонова музика для підкреслення атмосфери напруження. Також вороги мають свої звуки при атаках та пересуванні.

РОЗДІЛ 4
ТЕСТУВАННЯ

Тестування ігор, підмножина розробки ігор, є тестування програмного забезпечення процес контролю якості відеоігор. Основна функція тестування ігор – виявлення та документування програмних дефектів (також званих помилками). Тестування інтерактивного розважального програмного забезпечення – це високотехнологічна область, що потребує комп'ютерних знань, аналітичних навичок, навичок критичної оцінки та витривалості. В останні роки область тестування ігор зазнала критики за те, що вона надзвичайно напружена і невдячна як у фінансовому, так і в емоційному плані.

Нижче приведено популярні методи тестування ігор:

- Функціональне тестування. Тестери шукають загальні проблеми в грі або її інтерфейсі користувача та графіку, наприклад, проблеми з ігровою механікою, проблеми зі стабільністю та цілісністю ігрових активів. Тестування інтерфейсу користувача забезпечує зручність гри.
- Тестування на сумісність. Перевірка сумісності гри на різних пристроях і на різних конфігураціях обладнання та програмного забезпечення.
- Тестування продуктивності. Перевіряється загальна продуктивність гри. Налаштування продуктивності виконується для оптимізації швидкості гри.
- Тестування локалізації. Тестування локалізації стає надзвичайно важливим, коли гра орієнтована на глобальні ринки. Назви ігор, вміст і тексти потрібно перекладати та тестувати за допомогою пристроїв кількома мовами. Такі типи тестів можна виконати швидко (за допомогою хмарного доступу до пристроїв і автоматизації тестування).
- Тестування на замочування. Це тестування передбачає відтворення гри протягом тривалого періоду в різних режимах роботи.

- Тестування на відновлення. У програмному забезпеченні тестування відновлення перевіряє, наскільки добре програму можна відновитися після збоїв.
- Перевірка безпеки. Це робиться для того, щоб перевірити, наскільки безпечно працює програмне забезпечення від зовнішніх загроз. Захист даних від зовнішніх загроз, неконтрольованих обмежень доступу до системи, порушення даних, помилок операційної системи, систем зв’язку та слабких алгоритмів шифрування[22].

За допомогою функціонального тестування була перевірена створена відеогра. Критичних помилок не виявлено, функціонал працює як і планувалося. Створено набір тестових випадків, які наведено у додатку В.

ВИСНОВОК

В результаті виконання дипломної роботи було здійснене проектування, реалізація та тестування однокористувацької тривимірної комп'ютерної гри в жанрі roguelike. Був проведений збір та аналіз теоретичної інформації із галузі створення комп'ютерних ігор для визначення основних елементів, притаманних саме цьому жанру ігор. В якості ПЗ для створення гри було обрано ігровий рушій Unity3D. Програма складається із модулів, що дозволяються працювати із усіма необхідними компонентами (світло, звук, анімації, камера, графічні ефекти) та дозволяє створювати скрипти на мові програмування C#.

Перед реалізацією відеогри було проведено підготовчий етап:

- створено сюжет гри;
- створено макети кімнат гри;
- обумовлена логіка роботи генератора рівнів;
- обумовлена логіка роботи ШІ ворогів;
- був проведений пошук необхідних компонентів (моделі персонажів, елементи оточення, матеріали та текстури, звуки).

Для більш просунотого пересування ворогів був використаний плагін з відкритим кодом «A* Pathfinding Project Pro».

Результатами реалізації стало наступне:

- створено головне меню гри, з якого гравець може розпочати проходження гри;
- створений генератор рівнів, який створює унікальний за структурою рівень з заздалегідь заготовленого списку кімнат;
- присутні елементи користувацького інтерфейсу у вигляді шкали здоров'я персонажа гри .

В процесі тестування не було виявлено критичних помилок.

Головною метою даного ПЗ було створення розважального продукту, що дозволить користувачам цікаво провести час та відпочити. Реалізований ПП можна вважати завершеним та готовим для безкоштовного розповсюдження, оскільки дана робота в першу чергу була орієнтована на отримання практичних навичок в області створення комп’ютерних ігор.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Computerandvideogames [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.sciencedaily.com/terms/computer_and_video_games.htm

2. TheBeginnersGuidetoVideoGameDevelopment [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.gamedesigning.org/video-game-development/

3. Roguelikegamesdefinitioncoined [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.inverse.com/gaming/roguelike-games-definition-coined

4. HistoryoftheRoguelike, fromRoguetoHades[Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://screenrant.com/roguelike-definition-games-rogue-hades-roguelite-dungeon-crawler/

5. AnIntroductiontoRoguelikes [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу:https://www.gamingdigest.co/blog/an-intro-to-roguelikes/

6. DeadCells [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://deadcells.fandom.com/wiki/Dead_Cells

7. DeadCellsreview–theapotheosisoftheRoguelike [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://venturebeat.com/2018/08/06/dead-cells-review-the-apotheosis-of-the-roguelike/

8. DwarfFortressistheCraziestGameYou've (Probably) NeverHeardOf [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.ign.com/articles/2019/03/16/dwarf-fortress-steam-story

9. LearningtoloveDwarfFortress, gaming'sdeepestsimulation [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу:

https://www.eurogamer.net/articles/2017-04-16-how-to-actually-get-started-in-dwarf-fortress-gamings-grandest-sim

10. DarkestDungeon [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.darkestdungeon.com/game-overview/

11. DARKEST DUNGEON REVIEW [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.polygon.com/2016/2/24/11102056/darkest-dungeon-review-pc-mac

12. RogueLegacy [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://rogue-legacy.fandom.com/ru/wiki/Rogue_Legacy

13. 'RogueLegacy'– GrabYourSword andShield, CastleHam son'sOpenforBusiness! [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://toucharcade.com/2019/08/21/rogue-legacy-review/

14. LoopHeroreview [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.rockpapershotgun.com/loop-hero-review

15. LoopHeroreview: anunexpectedparableaboutparenting [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.polygon.com/reviews/22321086/loop-hero-review-retro-roguelike-pc-mac

16. TOP 10 GAME DEVELOPMENT ENGINES TODAY [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.youngwonks.com/blog/Top-10-Game-Development-Engines-Today

17. First-person (gaming) [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://ultimatepopculture.fandom.com/wiki/First-person_(gaming)

18. Процедурнагенераціяпідземель у roguelike[Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://habr.com/ru/post/354826/

19. Ігровийштучнийінтелект[Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим

доступу:https://uk.wikipedia.org/wiki/%D0%86%D0%B3%D1%80%D0%BE%D0%B2%D0%B8%D0%B9_%D1%88%D1%82%D1%83%D1%87%D0%BD%D0%B8%D0%B9_%D1%96%D0%BD%D1%82%D0%B5%D0%BB%D0%B5%D0%BA%D1%82

20. Пошук шляху [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу:https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%88%D1%83%D0%BA_%D1%88%D0%BB%D1%8F%D1%85%D1%83

21. АлгоритмпошукуA*[Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу:https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%BF%D0%BE%D1%88%D1%83%D0%BA%D1%83_A*

22. GameTesting: Types&HowtoTestMobile/DesktopApps[Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.guru99.com/game-testing-mobile-desktop-apps.html

ДОДАТОК А
ВИХІДНИЙ КОД ПРОГРАМИ

1. Скрипт CharacterController.cs

```
public abstract class CharacterController : MonoBehaviour
{
   #region VARIABLES
protected Animator animator;
   #endregion
   #region METHODS
protected void SetAnimationBool(intparam, bool value)
   {
animator.SetBool(param, value);
   }
protected bool GetAnimationBool(intparam)
   {
returnanimator.GetBool(param);
   }
protected void TriggerAnimation(intparam)
   {
animator.SetTrigger(param);
   }
   #endregion
   }
```

2. Скрипт EnemyController.cs

```
public class EnemyController : CharacterController
{
   #region VARIABLES

protectedAIPathaiPath;

protectedLayerMaskplayerLayer;
protectedGameObject player;

protected bool playerInSightRange;
protected bool playerInAttackRange;

protected float moveSpeed;
protected float rotationSpeed;
protected float sightRange;
protected float attackRange;

   #endregion

   #region PROPERTIES

protectedreadonlyintmoveParam = Animator.StringToHash("Move"); //bool
protectedreadonlyintattackParam = Animator.StringToHash("Attack"); //trigger
protectedreadonlyintdeathParam = Animator.StringToHash("Death"); //trigger

protected bool AttackInfo =>animator.GetCurrentAnimatorStateInfo(0).IsName("Attack");
protected bool DeathInfo =>animator.GetCurrentAnimatorStateInfo(0).IsName("Death");

   #endregion

   #region METHODS

protected void MoveTo(Vector3 movePosition)
   {
aiPath.destination = movePosition;
   }

protected bool CollisionDetection(float radius, LayerMasktargetLayer)
   {
returnPhysics.CheckSphere(transform.position, radius, targetLayer);
```

```
    }

protected void ChaseTarget(GameObject target)
    {
if (target == null) return;

vartargetPosition = target.transform.position;
MoveTo(targetPosition);
    }

protected void Attack(GameObject target)
    {
transform.LookAt(target.transform.position);
TriggerAnimation(attackParam);
    }

public void Death()
    {
TriggerAnimation(deathParam);
    }

    //DEBUGGING
protected void OnDrawGizmos()
    {
        //chase distance
Gizmos.color = Color.yellow;
Gizmos.DrawWireSphere(transform.position, sightRange);

        //attack distance
Gizmos.color = Color.red;
Gizmos.DrawWireSphere(transform.position, attackRange);
    }

    #endregion
}
```

3. СкриптPlayerController.cs

```
public class PlayerController : CharacterController
{
    #region VARIABLES

    [SerializeField] private PlayerData data;

private Vector3 _movementDirection;

    //data
private float _currentHealth;
private float _moveSpeed;
private float _rotationSpeed;
private float _invulnerabilityTime;
private float _invulnerabilityTimeStamp;

    #endregion

    #region PROPERTIES

privatereadonlyint _moveParam = Animator.StringToHash("Move"); //bool
privatereadonlyint _takeDamageParam = Animator.StringToHash("TakeDamage"); //trigger
privatereadonlyint _deathParam = Animator.StringToHash("Death"); //trigger

private bool GetHitInfo =>animator.GetCurrentAnimatorStateInfo(0).IsName("GetHit");
private bool DeathInfo =>animator.GetCurrentAnimatorStateInfo(0).IsName("Death");

    #endregion

    #region METHODS

private void LogicUpdate()
    {
if (DeathInfo || GetHitInfo)
return;
```

```
HandleInputs();
HandleAnimations();

Move(_movementDirection, _moveSpeed, _rotationSpeed);
   }

private void HandleInputs()
   {
      _movementDirection.x = Input.GetAxisRaw("Horizontal");
      _movementDirection.z = Input.GetAxisRaw("Vertical");
   }

private void HandleAnimations()
   {
SetAnimationBool(_moveParam, _movementDirection != Vector3.zero);
SetAnimationBool(_deathParam, _currentHealth<= 0);
   }

private void InitializeData()
   {
      _currentHealth = data.health;
      _moveSpeed = data.moveSpeed;
      _rotationSpeed = data.rotationSpeed;
      _invulnerabilityTime = data.invulnerabilityTime;
   }

private void Move(Vector3 movementDirection, float moveSpeed, float rotationSpeed)
   {
var velocity = movementDirection.normalized * (moveSpeed * Time.deltaTime);
transform.Translate(velocity, Space.World);

if (movementDirection == Vector3.zero) return;
vartoRotation = Quaternion.LookRotation(movementDirection, Vector3.up);
transform.rotation =
Quaternion.RotateTowards(transform.rotation, toRotation, rotationSpeed * Time.deltaTime);
   }

public void TakeDamage()
   {
if (_invulnerabilityTimeStamp>Time.time)
return;

      _currentHealth -= 1;
if (_currentHealth<= 0)
      {
Death();
      }

TriggerAnimation(_takeDamageParam);

      _invulnerabilityTimeStamp = Time.time + _invulnerabilityTime;
   }

private void Death()
   {
TriggerAnimation(_deathParam);
   }

   #endregion

   #region MonoBehaviour Callbacks

private void Start()
   {
animator = GetComponent<Animator>();

InitializeData();
   }

private void Update()
   {
LogicUpdate();
```

```
    }

    #endregion
}
```

4. СкриптSkeletonController.cs

```
public class SkeletonController : EnemyController
{
    #region VARIABLES

    [SerializeField] private EnemyData data;
    [SerializeField] private GameObject sword;

private Collider _swordCollider;

    #endregion

    #region METHODS

private void InitializeData()
    {
aiPath.maxSpeed = data.moveSpeed;
aiPath.rotationSpeed = data.rotationSpeed;

sightRange = data.sightRange;
attackRange = data.attackRange;
    }

private void HandleAnimations()
    {
SetAnimationBool(moveParam, !aiPath.reachedDestination);
    }

private void LogicUpdate()
    {
if (DeathInfo)
return;

if (AttackInfo)
        {
            _swordCollider.enabled = true;
return;
        }

HandleAnimations();

        _swordCollider.enabled = false;
playerInSightRange = CollisionDetection(sightRange, playerLayer);
playerInAttackRange = CollisionDetection(attackRange, playerLayer);

if (playerInSightRange&&playerInAttackRange) //attack
        {
MoveTo(transform.position);
Attack(player);
        }
if(!playerInSightRange&& !playerInAttackRange) //idle
        {
MoveTo(transform.position);
        }
if (playerInSightRange&& !playerInAttackRange) //chase
        {
ChaseTarget(player);
        }
    }

    #endregion

    #region MonoBehaviour Callbacks

private void Start()
    {
animator = GetComponent<Animator>();
```

```
aiPath = GetComponent<AIPath>();

      _swordCollider = sword.GetComponentInChildren<MeshCollider>();

player = GameObject.FindGameObjectWithTag("Player");
playerLayer = LayerMask.GetMask("Player");

InitializeData();
   }

private void Update()
   {
LogicUpdate();
   }

   #endregion
}
```

5. СкриптMenuController.cs

```
public class MenuController : MonoBehaviour
{
public void PlayPressed()
   {
SceneManager.LoadScene("Game");
   }

public void ExitPressed()
   {
Application.Quit();
Debug.Log("Exit pressed!");
   }
}
```

6. СкриптDoorway.cs

```
public class Doorway : MonoBehaviour
{
voidOnDrawGizmos()
   {
      Ray ray = new Ray(transform.position,
transform.rotation * Vector3.forward);

Gizmos.color = Color.red;
Gizmos.DrawRay(ray);
   }
}
```

7. СкриптRoom.cs

```
public class Room : MonoBehaviour
{
public Doorway[] doorways;
publicMeshCollidermeshCollider;

public Bounds RoomBounds
   {
get { return meshCollider.bounds; }
   }
}
```

8. СкриптStartRoom.cs

```
public class StartRoom : Room
{}
```

9. СкриптStartRoom.cs

```
publicclassEndRoom : Room
{}
```

10.СкриптLevelBuilder.cs

```
public class LevelBuilder : MonoBehaviour
```

```
{
public Room startRoomPrefab, endRoomPrefab;
public List<Room>roomPrefabs = new List<Room>();
public Vector2 iterationRange = new Vector2(3, 10);

private List<Doorway>availableDoorways = new List<Doorway>();

privateStartRoomstartRoom;
privateEndRoomendRoom;
private List<Room>placedRooms = new List<Room>();

privateLayerMaskroomLayerMask;

private void Start()
   {
roomLayerMask = LayerMask.GetMask("Room");
StartCoroutine(nameof(GenerateLevel));
   }

IEnumeratorGenerateLevel()
   {
WaitForSeconds startup = new WaitForSeconds(1);
WaitForFixedUpdate interval = new WaitForFixedUpdate();

yield return startup;

      //Place start room
PlaceStartRoom();
yield return interval;

      //Random iterations
int iterations = Random.Range((int) iterationRange.x, (int) iterationRange.y);

for (inti = 0; i< iterations; i++)
      {
         //Place random room from the list
PlaceRoom();
yield return interval;
      }

      //Place end room
PlaceEndRoom();
yield return interval;

      //Level generator finished
Debug.Log("Level generator finished");

      //Reset level generator
      //yield return new WaitForSeconds(3);

      //ResetLevelGenerator();
   }

private void PlaceStartRoom()
   {
      //Instantiate start room
startRoom = Instantiate(startRoomPrefab) as StartRoom;
startRoom.transform.parent = this.transform;

      //Get doorways from current room and add them randomly to
      //the list of available doorways
AddDoorwaysToList(startRoom, ref availableDoorways);

      //Position room
startRoom.transform.position = Vector3.zero;
startRoom.transform.rotation = Quaternion.identity;
   }

private void AddDoorwaysToList(Room room, ref List<Doorway> list)
   {
foreach (Doorway doorway in room.doorways)
      {
```

```
intrandDoor = Random.Range(0, list.Count);
list.Insert(randDoor, doorway);
        }
    }

private void PlaceRoom()
    {
        //Instantiate room
        Room currentRoom = Instantiate(roomPrefabs[Random.Range(0, roomPrefabs.Count)]) as Room;
currentRoom.transform.parent = this.transform;

        //Create doorway lists to loop over
        List<Doorway>allAvailableDoorways = new List<Doorway>(availableDoorways);
        List<Doorway>currentRoomDoorways = new List<Doorway>();
AddDoorwaysToList(currentRoom, ref currentRoomDoorways);

        //Get doorways from current room and add them randomly to
        //the list of available doorways
AddDoorwaysToList(currentRoom, ref availableDoorways);

boolroomPlaced = false;

        //Try all available doorways
foreach (Doorway availableDoorway in allAvailableDoorways)
        {
            //Try all available doorways in current room
foreach (Doorway currentDoorway in currentRoomDoorways)
            {
PositionRoomAtDoorway(ref currentRoom, currentDoorway, availableDoorway);
Physics.SyncTransforms();

                //Check room overlaps
if (CheckRoomOverlap(currentRoom))
                {
continue;
                }

roomPlaced = true;

                //Add room to list
placedRooms.Add(currentRoom);

                //Remove occupied doorways
currentDoorway.gameObject.SetActive(false);
availableDoorways.Remove(currentDoorway);

availableDoorway.gameObject.SetActive(false);
availableDoorways.Remove(availableDoorway);

                //Exit loop if room has been placed
break;
            }

            //Exit loop if room has been placed
if (roomPlaced)
            {
break;
            }
        }

        //Room couldn't be placed. Restart generator and try again
if (!roomPlaced)
        {
Destroy(currentRoom.gameObject);
ResetLevelGenerator();
        }
    }

private void PositionRoomAtDoorway(ref Room room, Doorway roomDoorway, Doorway targetDoorway)
    {
        //Reset room position and rotation
room.transform.position = Vector3.zero;
```

```
room.transform.rotation = Quaternion.identity;

        //Rotate room to match previous doorway orientation
        Vector3 targetDoorwayEuler = targetDoorway.transform.eulerAngles;
        Vector3 roomDoorwayEuler = roomDoorway.transform.eulerAngles;
floatdeltaAngle = Mathf.DeltaAngle(roomDoorwayEuler.y, targetDoorwayEuler.y);
        Quaternion currentRoomTargetRotation = Quaternion.AngleAxis(deltaAngle, Vector3.up);
room.transform.rotation = currentRoomTargetRotation * Quaternion.Euler(0, 180f, 0);

        //Position room
        Vector3 roomPositionOffset = roomDoorway.transform.position - room.transform.position;
room.transform.position = targetDoorway.transform.position - roomPositionOffset;

    }

private bool CheckRoomOverlap(Room room)
    {
        Bounds bounds = room.RoomBounds;
        //bounds.Expand(-0.1f);

Collider[] colliders = Physics.OverlapBox(bounds.center, bounds.size / 2,
room.transform.rotation, roomLayerMask);

if (colliders.Length> 0)
        {
            //Ignore collisions with current room
foreach (Collider c in colliders)
            {
if (c.transform.parent.gameObject.Equals(room.gameObject))
                {
continue;
                }
else
                {
Debug.LogError("Overlap detected");
return true;
                }
            }
        }

return false;
    }

private void PlaceEndRoom()
    {
        //Instantiate end room
endRoom = Instantiate(endRoomPrefab) as EndRoom;
endRoom.transform.parent = this.transform;

        //Create doorway lists to loop over
        List<Doorway>allAvailableDoorways = new List<Doorway>(availableDoorways);
        Doorway doorway = endRoom.doorways[0];

boolroomPlaced = false;

        //Try all available doorways
foreach (Doorway availableDoorway in allAvailableDoorways)
        {
            Room room = (Room) endRoom;
PositionRoomAtDoorway(ref room, doorway, availableDoorway);
Physics.SyncTransforms();

            //Check room overlaps
if (CheckRoomOverlap(endRoom))
            {
continue;
            }

roomPlaced = true;

            //Remove occupied doorways
doorway.gameObject.SetActive(false);
```

```
availableDoorways.Remove(doorway);

availableDoorway.gameObject.SetActive(false);
availableDoorways.Remove(availableDoorway);

            //Exit loop if room has been placed
break;
        }

        //Room couldn't be placed. Restart generator and try again
if (!roomPlaced)
        {
            //Destroy(endRoom.gameObject);
ResetLevelGenerator();
        }
    }

private void ResetLevelGenerator()
    {
Debug.LogError("Reset level generator");
StopCoroutine(nameof(GenerateLevel));

        //Delete all rooms
if (startRoom)
        {
Destroy(startRoom.gameObject);
        }

if (endRoom)
        {
Destroy(endRoom.gameObject);
        }

foreach (Room room in placedRooms)
        {
Destroy(room.gameObject);
        }

        //Clear lists
placedRooms.Clear();
availableDoorways.Clear();

        //Reset generator
StartCoroutine(nameof(GenerateLevel));
    }
}
```

11.СкриптData.cs

```
public class Data : ScriptableObject
{
    [Header("Basic data")]
public float moveSpeed = 5;
public float rotationSpeed = 360;
}
```

12.СкриптEnemyData.cs

```
public class EnemyData : Data
{
public float sightRange = 5;
public float attackRange = 1;
}
```

13.СкриптPlayerData.cs

```
public class PlayerData : Data
{
public float health = 3;
public float invulnerabilityTime = 2; //2 seconds after hit
}
```

ДОДАТОК В
ТЕСТОВІ СЦЕНАРІЇ

Таблиця В.1 – Тестування головного меню гри

Опис		Перевірка роботи кнопок меню
Передумови		Гра запущена
№	Дія	Очікуваний результат
1.	Натиснути кнопку «Розпочати гру»	Відбувається перехід до рівня гри
2.	Натиснути кнопку «Вихід»	Відбувається вихід із гри

Таблиця В.2– Тестування системи пошкоджень

Опис		Перевірка роботи системи пошкоджень
Передумови		Рівень запущено
№	Дія	Очікуваний результат
1.	Ворог атакує гравця	Кількість життів гравця зменшується

Таблиця В.3– Перевірка роботи тригера завершення рівня

Опис		Перевірка тригеру, що завершує рівень
Передумови		Гравець запустив гру та натиснув кнопку «Розпочати гру», гра завантажила рівень, пройшов рівень до кінцевої кімнати
№	Дія	Очікуваний результат
1.	Гравець зайшов у тригер кінця рівня	Рівень закінчився