МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЛЕСІ УКРАЇНКИ
Кафедра комп’ютерних наук та кібербезпеки

КИРИЧЕНКО ВАЛЕНТИН СЕРГІЙОВИЧ

# АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ РОЗРОБКИ КОМП'ЮТЕРНИХ ІГОР ЖАНРУ ROGUELIKE НА ПРИКЛАДІ «МАНДРІВКА ПІДЗЕМЕЛЛЯМИ»

Спеціальність: 122 Комп’ютерні науки
Освітньо-професійна програма: Комп’ютерні науки та інформаційні технології
Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр»

Науковий керівник:
БУЛАТЕЦЬКИЙ ВІТАЛІЙ ВІКТОРОВИЧ,
доцент кафедри комп’ютерних наук
та кібербезпеки

РЕКОМЕНДОВАНО ДО ЗАХИСТУ
Протокол №_____
засідання кафедри комп’ютерних наук
та кібербезпеки
від _____ 2025 р.
Завідувач кафедри

(_____) Гришанович Т. О.

ЛУЦЬК − 2025

ЗМІСТ

# ВСТУП

**Актуальність** теми в умовах стрімкого розвитку індустрії комп'ютерних ігор та зростаючої популярності незалежних (інді) проєктів, особливого значення набуває освоєння технологій та підходів до швидкої та ефективної розробки унікального ігрового контенту. Жанр RogueLike, з його акцентом на високій реіграбельності завдяки процедурній генерації рівнів та непередбачуваності ігрових ситуацій, залишається актуальним та вимогливим серед гравців. Розробка ігор у цьому жанрі дозволяє не лише створити захопливий інтерактивний продукт, але й є чудовим полігоном для відпрацювання ключових навичок сучасного розробника: від проектування складної логіки та алгоритмів (зокрема, процедурної генерації та штучного інтелекту) до роботи з сучасними ігровими рушіями. Використання потужного та гнучкого середовища розробки, такого як Unity, відкриває широкі можливості для реалізації амбітних ідей навіть для невеликих команд або окремих розробників. Таким чином, дослідження та практична реалізація процесу створення RogueLike гри на Unity є актуальним завданням, що дозволяє глибоко зануритись у специфіку інді-геймдеву та опанувати сучасні інструменти програмної інженерії в контексті ігрової розробки. Виконання даної роботи є особистим внеском автора у дослідження та практичне застосування сучасних підходів до розробки програмного забезпечення в ігровій індустрії.

**Метою** даної бакалаврської роботи є розробка функціонального прототипу RogueLike гри з реалізацією ключових механік жанру на базі ігрового рушія Unity.

Для досягнення поставленої мети було визначено наступні **завдання**:

- провести аналіз теоретичних основ, ключових елементів та особливостей жанру RogueLike, а також існуючих підходів до його реалізації;

- виконати огляд та аналіз аналогічних програмних розробок жанру RogueLike для виявлення їх переваг, недоліків та визначення вимог до власного програмного продукту;

- обґрунтувати вибір ігрового рушія Unity та інших інструментальних засобів, використаних у процесі розробки;

- спроектувати архітектуру програмного засобу, включаючи основні модулі, їх взаємодію та структури даних;

- реалізувати ключові ігрові механіки: процедурну генерацію рівнів, покрокову бойову систему, систему інвентарю, штучний інтелект супротивників та механіки взаємодії з ігровим світом;

- розробити користувацький інтерфейс та інтегрувати систему збереження та завантаження прогресу гри;

- провести тестування та налагодження розробленого програмного засобу для забезпечення його коректної роботи;

- скласти технічне завдання на розробку програмного засобу та керівництво користувачу.

**Об'єктом дослідження** є процес розробки комп'ютерних ігор жанру RogueLike.

**Предметом дослідження** є методи, алгоритми та інструментальні засоби реалізації ключових механік RogueLike гри в середовищі ігрового рушія Unity.

# РОЗДІЛ 1
# ОГЛЯД ТА АНАЛІЗ ЖАНРУ ROGUELIKE І ТЕХНОЛОГІЙ РОЗРОБКИ

## 1.1. Історія, ключові елементи та особливості жанру RogueLike

Жанр комп'ютерних ігор RogueLike має глибоке коріння, що сягає початку 1980-х років, і бере свою назву від культової гри Rogue, випущеної у 1980 році. Створена Кентом Кендаллом, Майклом Той та Ейтаном Циммерманом, гра Rogue заклала фундаментальні принципи, які визначили цілий піджанр role-playing games (RPG). Незважаючи на свій текстовий або псевдографічний інтерфейс, що був зумовлений технічними обмеженнями того часу, Rogue запропонувала гравцям захопливий та непередбачуваний досвід дослідження процедурно згенерованих підземель, повних небезпек та скарбів [8] . Успіх Rogue надихнув створення численних подібних ігор, які стали відомі як "roguelikes" – ігри, подібні до Rogue.

З часом сформувався набір визначальних характеристик, який став своєрідним "Берлінським Договором" жанру RogueLike, хоча його положення і можуть трактуватися дещо вільно в сучасних іграх [1]. До ключових елементів класичного RogueLike належать:

- процедурна генерація (Procedural Generation): Ігровий світ (рівні, розташування ворогів, предметів, пасток) генерується випадковим чином при кожному новому сеансі гри. Це забезпечує високу реіграбельність та унікальний досвід кожного разу [6];

- перманентна смерть (Permadeath): Смерть персонажа гравця є остаточною. Прогрес (окрім, можливо, розблокованого контенту в деяких сучасних варіаціях) втрачається, і гру потрібно починати з самого початку [1]. Цей елемент підвищує ставки та вимагає від гравця обережності та тактичного мислення;

- покроковість (Turn-based gameplay): Дії гравця та всіх ігрових сутностей (ворогів, ефектів) відбуваються покроково. Гравець має необмежений час на прийняття рішення перед виконанням свого ходу [12];

- сіткова основа (Grid-based movement): Ігровий світ зазвичай представлений сіткою (тайлами), і переміщення персонажів та об'єктів відбувається по цій сітці [8];

- комплексність (Complexity): Взаємодія між різними ігровими елементами (предмети, вміння, вороги, оточення) створює глибоку та непередбачувану синергію [1];

- управління ресурсами (Resource Management): Гравцеві необхідно ефективно управляти обмеженими ресурсами (здоров'я, мана, їжа, предмети) для виживання та просування;

- неідентифіковані предмети (Identification): Властивості знайдених предметів (зілля, сувої) можуть бути невідомі гравцеві до моменту їх використання або ідентифікації, що додає елемент ризику та експерименту.

З часом жанр еволюціонував, і з'явилися ігри, які запозичують певні елементи RogueLike, але відступають від деяких класичних принципів, особливо перманентної смерті або покроковості. Такі ігри часто називають RogueLite. У RogueLite іграх частина прогресу може зберігатися між спробами (наприклад, розблоковані здібності, предмети, покращення бази), що робить їх менш складними та більш доступними для широкої аудиторії.

Принципи геймдизайну в жанрі RogueLike зосереджені на створенні збалансованого поєднання випадковості та можливостей для стратегічного планування [6]. Процедурна генерація забезпечує несподіванку та унікальність кожного проходження, тоді як покроковість та комплексність механік дають гравцеві простір для обдуманих тактичних рішень. Дизайн має стимулювати адаптацію гравця до мінливих умов, експериментування з предметами та вміннями, а також навчитись приймати наслідки своїх дій в умовах перманентної смерті.

Таким чином, розуміння історії та фундаментальних принципів жанру RogueLike є критично важливим для успішної розробки власної гри, яка б відповідала очікуванням гравців та зберігала дух класичних представників жанру, водночас впроваджуючи сучасні ігрові механіки та технології.

## 1.2. Огляд технологій та інструментів для розробки ігор

Розробка сучасних комп'ютерних ігор, особливо в умовах обмежених ресурсів незалежних (інді) розробників, вимагає ефективного використання спеціалізованих технологій та інструментальних засобів. Центральне місце серед них займають ігрові рушії (game engines) [5] – програмні комплекси, що надають розробникам готові інструменти та функціонал для створення ігрового світу, обробки графіки, фізики, звуку, анімації, реалізації ігрової логіки та взаємодії з користувачем. Використання ігрового рушія дозволяє значно прискорити процес розробки, уникнути необхідності написання базових систем "з нуля" та зосередитись безпосередньо на унікальних аспектах гри [9].

На сучасному ринку існують численні ігрові рушії, які різняться за своїми можливостями, ліцензійною політикою, зручністю використання та орієнтацією на певні платформи чи жанри. Серед найбільш популярних рушіїв, які активно використовуються інді-розробниками, можна виділити Unity, Unreal Engine, GameMaker Studio 2, Godot Engine [2]. Кожен з них має свої переваги та недоліки, що робить вибір рушія важливим етапом проєктування.

Unity є одним з лідерів на ринку, особливо для розробки мобільних та інді-ігор. Він підтримує розробку як 2D, так і 3D проєктів, використовує мову програмування C# і надає потужний візуальний редактор та велику екосистему ассетів (Unity Asset Store) [10]. Це робить його привабливим вибором для розробників, які шукають гнучкий та багатофункціональний інструмент.

Unreal Engine, розроблений Epic Games, відомий своїми передовими можливостями 3D графіки та використовує мову C++. Він є популярним вибором

для створення високобюджетних проєктів та ігор з реалістичною графікою, хоча також підтримує 2D розробку.

GameMaker Studio 2 часто обирають початківці завдяки його простому інтерфейсу та власному скриптовому мові GML, хоча він також підтримує C#. Він добре підходить для швидкої розробки 2D ігор.

Godot Engine є безкоштовним рушієм з відкритим кодом, який активно розвивається спільнотою. Він підтримує кілька мов програмування, включаючи GDScript (схожий на Python), C# та C++, і є хорошою альтернативою для розробників, які цінують свободу та прозорість.

Для розробки 2D ігор, які є типовими для багатьох класичних RogueLike проєктів, більшість сучасних рушіїв мають необхідні інструменти: системи спрайтової анімації, роботу з тайловими картами, 2D фізику. Зокрема, Unity має розвинений 2D пайплайн, що спрощує роботу з двовимірною графікою [10].

Процедурна генерація, як ключовий елемент RogueLike жанру, також може бути реалізована за допомогою різних рушіїв. Рушії надають можливості для створення та маніпулювання ігровими об'єктами під час виконання програми, що є основою для імплементації алгоритмів генерації контенту [6]. Можливості мов програмування, які підтримуються рушіями (наприклад, C# в Unity), дозволяють розробляти складні алгоритми процедурної генерації підземель, предметів, ворогів тощо.

Загалом, використання ігрового рушія порівняно з розробкою "з нуля" дає низку суттєвих переваг [9]:

- економія часу: Готові інструменти для графіки, фізики, звуку, введення/виведення даних;
- спрощення процесу: Інтуїтивно зрозумілі редактори та візуальні інструменти;
- кросплатформенність: Можливість випуску гри на різних платформах з мінімальними змінами;
- доступ до ресурсів: Великі спільноти, документація, навчальні матеріали та магазини ассетів.

Зважаючи на ці фактори, вибір ігрового рушія є критично важливим рішенням для успішної та ефективної розробки інді-гри, дозволяючи зосередитись на креативних та унікальних аспектах проєкту.

### 1.2.1. Unity: огляд та основні можливості

Unity [11] є інтегрованим середовищем розробки, призначеним для створення двовимірних (2D) та тривимірних (3D) інтерактивних застосунків, зокрема відеоігор. Започаткований у 2005 році, рушій швидко здобув популярність завдяки своїй доступності, гнучкості та орієнтації на мультиплатформенну розробку. Ключовою перевагою Unity є можливість створювати проєкти, які можуть бути розгорнуті на значній кількості цільових платформ, включаючи стаціонарні та мобільні операційні системи, ігрові консолі, веб-браузери та платформи віртуальної/доповненої реальності. Така універсальність робить Unity привабливим вибором як для великих студій, так і для незалежних розробників.

Архітектура Unity базується на компонентно-орієнтованому підході, де ігрові об'єкти (GameObjects) є контейнерами для компонентів, що визначають їхню поведінку, вигляд та взаємодію. Ядром інтегрованого середовища є потужний візуальний редактор (Unity Editor), який надає інструменти для дизайну рівнів, розміщення об'єктів, налаштування їх властивостей та візуалізації ігрового світу в режимі реального часу.

Для реалізації ігрової логіки в Unity переважно використовується мова програмування C# у поєднанні з фреймворком .NET. C# є об'єктно-орієнтованою мовою, яка добре інтегрована з можливостями рушія та має велику кількість бібліотек. Рушій надає багатий API (Application Programming Interface), що дозволяє програмно взаємодіяти з усіма аспектами гри – від об'єктів сцени та їхніх компонентів до систем фізики, рендерингу та введення користувача.

Серед основних функціональних систем, які пропонує Unity, слід виділити:

- система рендерингу: Забезпечує візуалізацію 2D та 3D графіки, підтримує різні пайплайни рендерингу (Built-in, URP, HDRP) для оптимізації та

досягнення потрібного візуального стилю. Для 2D розробки Unity має спеціалізовані інструменти для роботи зі спрайтами, анімацією спрайтів та тайловими картами;

- фізичний рушій: Імплементує закони фізики для реалістичної або стилізованої взаємодії об'єктів у 2D та 3D просторі;

- система анімації: Дозволяє створювати та управляти анімацією персонажів та об'єктів;

- система інтерфейсу користувача (UI): Надає інструменти для розробки елементів керування, інформаційних панелей та меню гри;

- аудіосистема: Забезпечує інтеграцію та управління звуковими ефектами та музичним супроводом;

- система ассетів та управління проєктом: Включає Unity Asset Store – онлайн-магазин ресурсів, а також систему пакетів (Package Manager) для інтеграції додаткових функціональних модулів та бібліотек.

Крім того, Unity включає інструменти для профілювання та оптимізації, що дозволяє аналізувати продуктивність гри та вносити необхідні покращення для ефективної роботи на цільових платформах. Гнучкість архітектури та широкий набір інструментів роблять Unity ефективним вибором для розробки ігор різної складності та жанрової приналежності.

### 1.2.2. Огляд алгоритмів процедурної генерації

Процедурна генерація контенту (Procedural Content Generation, PGC) є важливою технікою в розробці комп'ютерних ігор, особливо для жанрів, що вимагають високої реіграбельності та непередбачуваності, як-от RogueLike. Замість створення кожного ігрового рівня або елемента вручну, PGC використовує алгоритми для автоматичного генерування контенту під час виконання програми або на етапі попередньої підготовки. Це дозволяє створювати унікальні ігрові сесії при кожному новому запуску, забезпечуючи свіжий досвід для гравця. Існує багато підходів та алгоритмів для процедурної

генерації ігрових рівнів, зокрема підземель, які є типовими для RogueLike ігор. Розглянемо декілька поширених теоретичних моделей.

Одним з інтуїтивно зрозумілих методів є генерація на основі розміщення кімнат та прокладання коридорів (Room-Carving). Цей підхід передбачає випадкове розміщення прямокутних або інших форм кімнат на ігровій сітці. Після розміщення кімнат, алгоритм з'єднує їх між собою, прокладаючи коридори. Коридори можуть генеруватися, наприклад, за допомогою простого шляху від центру однієї кімнати до центру іншої, або ж використовуватись метод "п'яниці, що блукає" (Drunkard's Walk), коли "коридор" створюється випадковими кроками від однієї точки до іншої, вирізаючи простір у стінах. Перевагами цього методу є його відносна простота реалізації та можливість створення структурованих рівнів з чітко вираженими кімнатами. Недоліками можуть бути менша варіативність форм коридорів та потенційна складність уникнення ізольованих ділянок або надмірної кількості перетинів.

Іншим популярним підходом для генерації більш органічних структур, таких як печери, є використання клітинних автоматів (Cellular Automata). Цей метод базується на застосуванні простих правил до кожної клітинки сітки на основі стану її сусідів протягом декількох ітерацій. Починаючи з випадково заповненої сітки (наприклад, печера/стіна), правила визначають, чи повинна клітинка залишитися стіною чи стати частиною печери в наступному кроці. Поступове застосування правил призводить до формування зв'язних областей та природних обрисів печерних систем. Переваги клітинних автоматів – здатність створювати візуально привабливі, органічні структури. Складність полягає у підборі правил, які гарантують зв'язність усіх частин рівня та відсутність надто малих або ізольованих порожнин.

Двійкове розбиття простору (Binary Space Partitioning, BSP Trees) є рекурсивним алгоритмом, що дозволяє створювати ієрархічні структури рівня. Процес починається з великої області, яка потім рекурсивно ділиться на дві менші підобласті за допомогою розділяючої лінії або площини. Цей процес повторюється до досягнення певного рівня деталізації або мінімального розміру

областей. Після завершення розбиття в кожній фінальній області можуть розміщуватися кімнати, а сусідні області на одному рівні ієрархії з'єднуються коридорами або проходами. BSP-дерева добре підходять для створення рівнів з чіткою структурою та різними за розміром зонами. Недоліком може бути певна передбачуваність структури, оскільки вона базується на прямолінійних розбиттях.

Також для процедурної генерації можуть використовуватися шумові функції, такі як шум Перліна (Perlin Noise) або шум Сімплекса (Simplex Noise). Ці функції генерують псевдовипадкові, але плавні значення по координатах, які можна інтерпретувати як висоту, щільність або інші параметри. Застосовуючи поріг до значень шумової функції на сітці, можна створити візерунки, схожі на природні ландшафти або контури печер. Наприклад, клітинки, де значення шуму вище певного порогу, можуть стати підлогою, а нижче – стінами. Шумові функції дозволяють створювати дуже органічні та різноманітні форми. Однак, забезпечення зв'язності всіх частин рівня може вимагати додаткових кроків або використання комбінованих підходів

Вибір конкретного алгоритму процедурної генерації значною мірою залежить від бажаного візуального стилю рівня, необхідної структури (наприклад, лабіринт, система печер, набір кімнат) та вимог до ігрового процесу. Часто в розробці ігор використовуються комбіновані підходи, що поєднують елементи різних алгоритмів для досягнення більш цікавих та різноманітних результатів. Розуміння цих теоретичних основ є важливим кроком для успішної реалізації системи процедурної генерації в RogueLike грі.

### 1.2.3. Огляд штучного інтелекту у RogueLike іграх

Штучний інтелект (ШІ) відіграє важливу роль у створенні захопливого та динамічного ігрового досвіду, особливо в жанрах, де гравець взаємодіє з віртуальним світом, населеним неігровими персонажами та супротивниками. У

RogueLike іграх ШІ переважно зосереджений на управлінні поведінкою ворогів, роблячи їх передбачуваними на базовому рівні (відповідно до правил покроковості), але водночас здатними створювати тактичні виклики для гравця. На відміну від складних симуляцій поведінки у високобюджетних проєктах, ШІ в класичних RogueLike часто базується на простих, але ефективних алгоритмах, що відповідають покроковій та сітковій структурі гри.

Основними завданнями ШІ супротивників у RogueLike є: переміщення по ігровому світу, виявлення гравця або інших цілей, прийняття рішень щодо дій (атака, відступ, використання здібностей) та взаємодія з оточенням. Реалізація руху ШІ на сітковій карті зазвичай вимагає використання алгоритмів пошуку шляху. Класичні алгоритми, такі як A* (читається як "A-зірка"), часто застосовуються для знаходження оптимального або найкоротшого шляху від поточної позиції ворога до цілі (наприклад, позиції гравця), враховуючи перешкоди на карті (стіни, непрохідні об'єкти). Інші, простіші алгоритми можуть використовуватися для базового слідування або патрулювання.

Важливим аспектом ШІ є його сприйняття ігрового світу. У RogueLike це часто реалізується через механіку "поля зору" (Field of View, FoV). Алгоритми FoV обчислюють, які клітинки на карті є видимими для певної сутності з урахуванням блокуючих огляд перешкод. ШІ ворогів може використовувати інформацію про поле зору, щоб виявляти присутність гравця, визначати його місцезнаходження та реагувати відповідним чином – переходити в стан агресії, наближатися для атаки або уникати прямого контакту. Рішення, які приймає ШІ, можуть базуватися на простих правилах (наприклад, якщо гравець у полі зору та в межах досяжності, атакувати; інакше – рухатися до останньої відомої позиції гравця або патрулювати), або бути більш складними, враховуючи стан самого ворога (здоров'я, наявність здібностей) та тактичну ситуацію на полі бою.

## 1.3. Огляд та аналіз аналогічних програмних розробок

Для формування обґрунтованих вимог до власної розробки та виявлення найбільш успішних підходів до реалізації ігрових механік у жанрі, було проведено детальний порівняльний аналіз низки впливових ігор RogueLike та RogueLite. Цей аналіз охоплює різноманітні приклади, що демонструють спектр реалізацій жанрових особливостей та підходів до геймдизайну. Серед обраних для аналізу ігор – Brogue [3], що є визнаним представником "чистих" RogueLike, а також The Binding of Isaac [9], Hades [7] та Enter the Gungeon [4], які є яскравими прикладами сучасних RogueLite ігор (Оглянути дані ігри можна у Дотатку В). Метою порівняння є не лише опис характеристик кожної гри, а й виділення їхніх спільних рис, значних відмінностей, ключових переваг та недоліків у контексті жанру, а також винесення уроків для розробки власного проєкту.

Одним з фундаментальних елементів жанру є процедурна генерація ігрового світу, і її реалізація значно варіюється між аналізованими іграми. У Brogue, яка вважається еталоном класичного RogueLike, підземелля генеруються повністю випадковим чином. Це забезпечує максимальну непередбачуваність кожного проходження та унікальний досвід щоразу, оскільки розташування кімнат, коридорів, ворогів та предметів щоразу нове. Перевагою такого підходу є безпрецедентна реіграбельність, проте недоліком може бути відсутність ретельно продуманих "дизайнерських" моментів. На противагу цьому, The Binding of Isaac також активно використовує процедурну генерацію кімнат та їхнього наповнення, але її сила полягає у величезній кількості випадкових предметів, що можуть створювати потужні та часто непередбачувані синергії. Hades застосовує більш контрольований підхід: рівні складаються з фіксованих, але випадково скомпонованих кімнат, що забезпечує баланс між елементом випадковості та збереженням високої якості геймдизайну кожної окремої зони. Така реалізація дозволяє краще інтегрувати наратив та вибудовувати складніші бойові сценарії. Enter the Gungeon також генерує рівні процедурно, але з акцентом на створенні динамічних арен, що є ідеальним для її "bullet hell"

механік. Різноманіття цих підходів свідчить про те, що оптимальна стратегія процедурної генерації обирається залежно від бажаного ігрового досвіду та акцентів.

Перманентна смерть – ще одна невід'ємна характеристика жанру, однак її імплементація суттєво впливає на поріг входження та утримання гравців. У Brogue смерть персонажа є абсолютно остаточною, і весь накопичений прогрес втрачається, що підвищує ставки та вимагає від гравця виняткової обережності та тактичного мислення. Цей елемент є перевагою для тих, хто шукає максимальний виклик, але може бути значним недоліком для нових гравців. Натомість, The Binding of Isaac, Hades та Enter the Gungeon належать до піджанру RogueLite, де перманентна смерть інтегрована із системою мета-прогресії. Це означає, що після загибелі гравець втрачає поточний прогрес забігу, але розблоковує нові здібності, персонажів чи предмети, які стануть доступними в майбутніх спробах. У Hades смерть навіть інтегрована в сюжетний наратив, що робить її менш фруструючою. Такий підхід значно знижує поріг входження, зменшує фрустрацію та мотивує гравця до подальших спроб, забезпечуючи довготривале залучення, хоча й може дещо зменшити складність у порівнянні з класичними RogueLike.

Бойова система є ще одним ключовим аспектом, що значно відрізняє аналізовані ігри та впливає на їхній темп та вимоги до гравця. Brogue використовує класичну покрокову бойову систему на сітці, де гравець має необмежений час на обмірковування кожного ходу, що робить акцент на глибокому тактичному плануванні та використанні елементів оточення. Перевага цього підходу – у можливості ретельно продумувати кожне рішення. На противагу цьому, The Binding of Isaac, Hades та Enter the Gungeon пропонують бойову систему в реальному часі. Isaac є швидким top-down шутером, який вимагає від гравця постійного маневрування та ухиляння від снарядів. Hades – це динамічний hack-and-slash з акцентом на швидкі атаки, ривки та комбінації здібностей. Enter the Gungeon є повноцінним "bullet hell" шутером, де гравець повинен постійно ухилятися від величезної кількості снарядів, що вимагає

високої реакції та координації. Ці ігри пропонують більш динамічний та інтенсивний геймплей, що є перевагою для гравців, які шукають екшн, але може бути недоліком для тих, хто віддає перевагу більш розміреній тактиці.

З точки зору презентації та використання технологій, ігри також демонструють значні відмінності. Brogue залишається вірною аскетичному ASCII-інтерфейсу, що, незважаючи на свою функціональність та чітке відображення ігрової інформації, може бути візуальним недоліком для сучасної аудиторії. The Binding of Isaac використовує стилізовану піксельну графіку, яка, хоч і є впізнаваною, не всім до вподоби. Hades вирізняється високоякісною, деталізованою hand-drawn графікою, плавними анімаціями, професійною озвучкою та глибокою інтеграцією сюжету. Enter the Gungeon також використовує деталізований піксель-арт, але з дуже динамічною анімацією та візуальними ефектами, що створюють яскравий та насичений візуальний стиль. Важливо підкреслити, як ігрові рушії сприяли цим реалізаціям. Enter the Gungeon була розроблена на Unity, і рушій забезпечив реалізацію складних систем: від ефективної 2D фізики для інтенсивних перестрілок до можливостей динамічного створення та маніпулювання ігровими об'єктами для процедурної генерації. Гнучкий скриптинг на C# в Unity дозволив створити величезну кількість унікальної зброї та предметів, а мультиплатформенність рушія забезпечила широке охоплення аудиторії. Це підтверджує, що Unity є потужним та гнучким інструментом для розробки сучасних RogueLite ігор, здатним забезпечити реалізацію як базових механік жанру, так і складних, динамічних систем.

Підсумовуючи, порівняльний аналіз показав, що жанр RogueLike/Lite має значну гнучкість у реалізації ключових елементів, дозволяючи розробникам адаптувати їх під різні геймплейні цілі та художні стилі. Переваги та недоліки кожного підходу – від складної покроковості Brogue до динамічних RogueLite ігор з глибокою мета-прогресією, таких як Isaac, Hades та Gungeon – дали цінне розуміння для формування вимог до власного програмного продукту. Це дозволило обрати найбільш релевантні механіки та технічні рішення з метою створення захопливої та технічно грамотної RogueLike гри на Unity.

# РОЗДІЛ 2
# АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ РОЗРОБКИ КОМП'ЮТЕРНИХ ІГОР ЖАНРУ ROGUELIKE НА ПРИКЛАДІ «МАНДРІВКА ПІДЗЕМЕЛЛЯМИ»

## 2.1. Постановка задачі, призначення та вимоги до програмного засобу «Мандрівка Підземеллями»

**Постановка задачі**

В умовах актуальності жанру RogueLike та наявності сучасних ігрових рушіїв, які дозволяють ефективно реалізовувати комплексні ігрові механіки, виникає задача розробки програмного забезпечення, що імплементує ключові принципи даного жанру. В рамках даної бакалаврської роботи поставлена задача розробки функціонального прототипу комп'ютерної гри в жанрі RogueLike на базі ігрового рушія Unity. Головним завданням є реалізація базового ігрового циклу, що включає процедурну генерацію ігрового світу, управління персонажем в покроковому режимі, взаємодію з оточенням, систему бою та інвентарю. Результатом роботи має стати програмний засіб, що демонструє працездатність реалізованих механік та слугує основою для подальшого розвитку проєкту.

Призначення програмного засобу «Мандрівка Підземеллями» полягає у наданні користувачеві інтерактивного досвіду дослідження процедурно згенерованих підземель, боротьби з монстрами, пошуку предметів та розвитку персонажа в умовах покрокового ігрового процесу та перманентної смерті. Програмний засіб орієнтований на користувачів, які цінують високу реіграбельність, тактичну глибину та непередбачуваність, властиву жанру RogueLike.

**Вимоги до функціональних характеристик**

Програмний засіб має забезпечувати систему процедурної генерації підземель, яка при кожному новому проходженні створюватиме унікальну структуру рівня, включаючи розташування кімнат, коридорів, сходів на

наступний рівень та ключових ігрових об'єктів, таких як гравець, вороги та предмети. Основний ігровий процес має базуватися на покроковій системі руху та взаємодії, де дії всіх сутностей на ігровому полі виконуються послідовно, надаючи гравцеві час для прийняття тактичних рішень. Невід'ємною частиною функціоналу є бойова система, що дозволяє персонажам (гравцеві та ворогам) атакувати одне одного з розрахунком шкоди на основі їхніх характеристик.

Система ігрових сутностей повинна включати гравця та різні типи ворогів, кожен з яких має визначений набір характеристик, таких як показники здоров'я, сили атаки та захисту. Для ворогів має бути реалізовано базовий штучний інтелект (AI), який визначає їхню поведінку у відповідь на дії гравця та переміщення по згенерованому рівню. Важливою механікою є система інвентарю, яка надає гравцеві можливість підбирати знайдені предмети, зберігати їх та використовувати у відповідний момент гри. Додатково, має бути реалізовано систему предметів, що включає різні типи інтерактивних об'єктів, таких як споживані зілля для відновлення здоров'я або сувої з магічними ефектами, кожен з яких має свої унікальні властивості та вплив на ігровий процес.

Для створення атмосфери дослідження та обмеження інформації про нерозвідані ділянки рівня необхідно впровадити систему "туману війни" (Fog of War) та поля зору (Field of View), що відображатиме для гравця лише видиму область навколо його персонажа. Програмний засіб повинен коректно обробляти перехід між різними рівнями підземелля, забезпечуючи безперервність ігрового досвіду. Також має бути визначена та реалізована умова завершення гри (наприклад, при досягненні персонажем нульового рівня здоров'я). Для зручності гравця повинна бути імплементована система збереження та завантаження поточного прогресу гри.

**Вимоги до інтерфейсу користувача**

Програмний засіб повинен забезпечувати візуальне представлення ігрового світу, включаючи ігрове поле з підземеллям, персонажами та об'єктами, використовуючи графічні елементи, такі як спрайти або тайли. Інтерфейс

користувача має надавати чітке відображення стану гравця, включаючи показники здоров'я, поточний рівень та досвід. Необхідно реалізувати візуальний доступ до інвентарю для зручного перегляду наявних предметів та можливості їх використання. Важливою частиною інтерфейсу є лог повідомлень, який інформує гравця про всі значущі події під час гри, такі як результати бойових дій, знайдені скарби або спрацьовані ефекти. Також має бути присутнє головне меню, що надає користувачеві стандартні опції, такі як початок нової гри, можливість завантажити раніше збережену гру, зберегти поточний прогрес та вийти з гри.

## 2.2. Вибір моделі розробки програмного засобу «Мандрівка Підземеллями»

Розробка програмного забезпечення є складним процесом, що складається з низки взаємопов'язаних етапів, які спільно утворюють життєвий цикл програмного забезпечення (ЖЦПЗ). ЖЦПЗ охоплює період від моменту виникнення ідеї або потреби у програмному продукті до його вилучення з експлуатації. Стандарт міжнародної організації ISO/IEC 12207:1995 «Information Technology – Software Life Cycle Processes» визначає структуру життєвого циклу, що включає основні процеси, дії та задачі, необхідні для створення та супроводу програмного забезпечення. Типові етапи ЖЦПЗ включають: формування та аналіз вимог, проектування, реалізацію (кодування), тестування, впровадження (розгортання) та супровід (підтримку та розвиток).

Існує кілька моделей ЖЦПЗ, які пропонують різні підходи до організації цих етапів. Серед найбільш відомих можна виділити каскадну (водоспадну) модель, спіральну модель, ітеративну модель та гнучкі (Agile) методології, такі як Scrum [19]. Кожна модель має свої переваги та недоліки і застосовується залежно від специфіки проєкту, вимог, ресурсів та термінів.

Для розробки програмного засобу «Мандрівка Підземеллями», враховуючи особливості створення ігрових прототипів, потенційну еволюцію

вимог у процесі розробки та обмеженість ресурсів в рамках бакалаврської роботи, було обрано ітеративну модель розробки (Iterative Model). Ця модель передбачає розбивку всього проєкту на низку коротких ітерацій, кожна з яких включає міні-етапи аналізу, проектування, реалізації та тестування. Результатом кожної ітерації є працездатний фрагмент програмного продукту або його покращена версія.

Обґрунтування вибору ітеративної моделі для розробки RogueLike гри на Unity полягає у наступному:

- гнучкість: Ітеративний підхід дозволяє легко адаптуватися до змін у вимогах або вносити корективи до ігрових механік на основі результатів тестування та власного досвіду розробки. У процесі створення гри часто виникає потреба в експериментах з балансом, відчуттям гри, що легше реалізувати при ітеративному підході;

- раннє отримання робочого продукту: Вже після перших ітерацій з'являється базовий прототип з реалізованими ключовими механіками , який можна тестувати та демонструвати. Це дозволяє отримати зворотний зв'язок та переконатись у правильності обраних рішень на ранніх етапах;

- управління ризиками: Ітеративна модель дозволяє виявляти та вирішувати основні ризики проєкту (наприклад, складність реалізації процедурної генерації або AI) на початку роботи, а не на фінальних етапах;

- відповідність характеру проєкту: Розробка ігрового прототипу часто має дослідницький характер. Ітерації дозволяють послідовно додавати функціонал, тестувати його ефективність та поступово нарощувати складність системи.

Кожна ітерація розробки програмного засобу «Мандрівка Підземеллями» включала: уточнення частини вимог, проектування конкретних модулів (наприклад, системи інвентарю або нових типів ворогів), їх програмну реалізацію на Unity з використанням C#, тестування реалізованого функціоналу та інтеграцію з існуючим кодом. Такий підхід дозволив ефективно організувати процес розробки в рамках визначених термінів бакалаврської роботи та отримати функціональний прототип гри.

## 2.3. Загальний опис проєкту «Мандрівка Підземеллями»

Розробка програмного засобу «Мандрівка Підземеллями» здійснена на базі ігрового рушія Unity з використанням мови програмування C#, що дозволило застосувати сучасні підходи до проектування та реалізації програмних систем. Архітектура проєкту побудована за принципами об'єктно-орієнтованого програмування та активно використовує компонентно-орієнтовану парадигму, яка є фундаментальною для розробки в середовищі Unity. Це забезпечило високу гнучкість, модульність та можливість ефективного управління складною ігровою логікою.

Загальна архітектура програмного засобу може бути представлена як сукупність взаємодіючих модулів та ключових класів, які відповідають за різні аспекти ігрового процесу:

Управління грою та ігровим циклом. Центральним елементом, що координує роботу всіх підсистем, є менеджер гри (клас GameManager). Він відповідає за ініціалізацію гри, запуск та завершення рівнів, обробку покроковості ігрового процесу, управління станами гри (головне меню, ігровий процес, пауза) та перевірку умов перемоги або поразки гравця. GameManager взаємодіє з іншими менеджерами та сутностями, делегуючи їм виконання відповідних завдань у межах ігрового циклу.

Управління ігровим світом та процедурна генерація. Модуль управління картою (клас MapManager) відповідає за представлення ігрового світу – підземелля. Основною функцією цього модуля є процедурна генерація рівнів. З використанням спеціалізованого алгоритму клас ProcGen створює унікальну структуру підземелля при кожному новому проходженні, визначаючи розташування кімнат, коридорів, стін та проходів. Дані про кожен елемент карти зберігаються в об'єктах TileData, що містять інформацію про позицію,

прохідність, стан видимості тощо. MapManager також відповідає за розміщення на згенерованій карті всіх ігрових об'єктів – гравця, ворогів, предметів, сходів. Сутності звертаються до MapManager для отримання інформації про властивості тайлів та наявності на них інших об'єктів.

Система сутностей та їхня взаємодія. Ядро ігрового процесу формується системою сутностей, яка реалізована на базі компонентно-орієнтованого підходу Unity. Основні класи, що представляють ігрові сутності та їхні ключові компоненти, проілюстровано на рисунку 2.1.

Як видно з діаграми(Рисунок 2.1), базовим для всіх ігрових об'єктів є клас Entity, який містить спільні властивості (позиція, назва, візуальне представлення) та базові методи (рух, взаємодія). Клас Player, що представляє головного героя, успадковує від Entity та додає специфічну для гравця логіку (обробка введення, розвиток). Бойові характеристики та логіка бою для сутностей інкапсульовані в компоненті Fighter, а управління предметами – в компоненті Inventory. Ці компоненти прикріплюються до об'єктів Entity, надаючи їм відповідний функціонал.

Система штучного інтелекту для супротивників реалізована через ієрархію класів, що базується на абстрактному класі AI. Конкретні типи ворогів, такі як HostileEnemy та ConfusedEnemy, успадковують від AI та реалізують власну логіку прийняття рішень та виконання дій. AI-контролери взаємодіють з об'єктами Entity, якими вони керують, викликаючи їхні методи руху та бою. Оглянути графічний вигляд проєкту можна у Додатку Г.

Інтерфейс користувача. Для взаємодії гравця з грою розроблено інтерфейс користувача (UI) з використанням стандартних засобів Unity UI. UIManager відповідає за відображення ігрового поля, панелей зі статистикою гравця (здоров'я, рівень), вікна інвентарю та логу повідомлень, що інформує гравця про події. Також реалізовано головне меню гри.

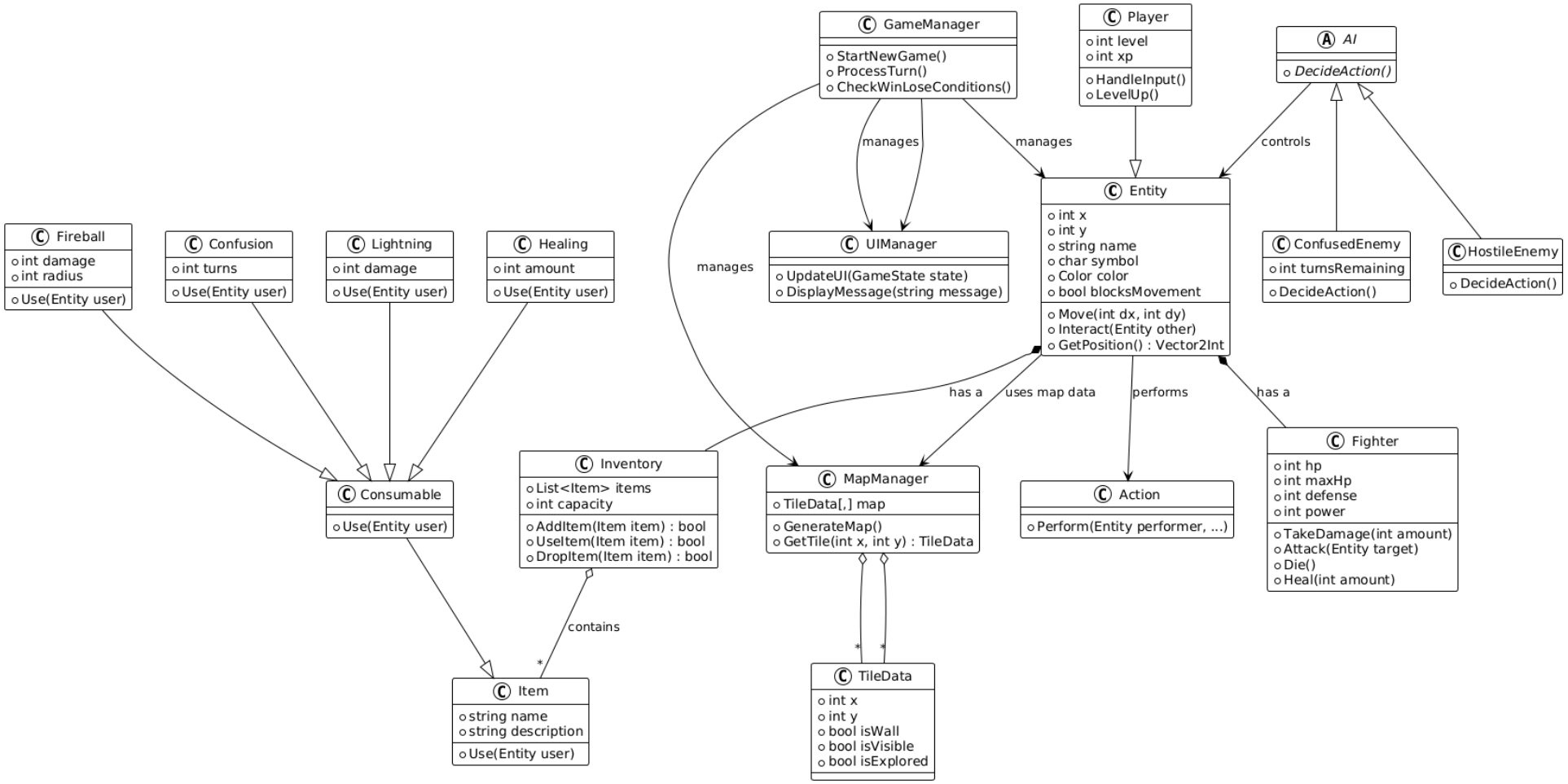



Рисунок 2.1 – Діаграма класів основних сутностей та компонентів програмного засобу «Мандрівка Підземеллями»

Система предметів та інвентарю. Взаємодія з предметами є важливою частиною RogueLike геймплею. Система предметів базується на класі Item, який є предком для різних типів предметів, включаючи споживані предмети (Consumable) та їхні конкретні реалізації (зілля здоров'я, сувої магії). Компонент Inventory у сутностей дозволяє збирати, зберігати та використовувати ці предмети. Використання предметів реалізовано через виклик відповідних методів, що застосовують їх ефекти до сутності, яка їх використовує.

Система збереження та завантаження. Для забезпечення можливості збереження прогресу гри реалізовано систему збереження/завантаження (клас SaveManager). Ця система відповідає за серіалізацію та десеріалізацію стану ключових ігрових об'єктів та даних для подальшого відновлення ігрової сесії. Використання додаткових бібліотек, таких як OdinSerializer, дозволило ефективно працювати зі складними структурами даних при збереженні.

Загалом, архітектура програмного засобу «Мандрівка Підземеллями» є типовою для ігор, розроблених на Unity, базується на принципах модульності, компонентності та об'єктно-орієнтованого підходу. Це дозволило реалізувати комплекс взаємодіючих систем, необхідних для функціонування RogueLike гри з процедурною генерацією контенту та покроковою механікою.

### 2.3.1. Математична модель та алгоритми використані у проєкті «Мандрівка Підземеллями»

Математична модель процесів та алгоритмічна основа програмного засобу «Мандрівка Підземеллями» формуються ключовими алгоритмами, що забезпечують генерацію ігрового світу, рух та взаємодію сутностей.

Центральним елементом математичної моделі є алгоритм процедурної генерації підземелля, реалізований у класі ProcGen.cs. Цей алгоритм базується на методі генерації розміщенням кімнат з'єднаних коридорами. Ключовою структурою даних, що використовується в цьому алгоритмі для представлення окремих кімнат, є клас RectangularRoom. Цей клас інкапсулює інформацію про

прямокутну область на карті, визначаючи її позицію (x, y) та розміри (width, height). Фрагмент коду класу RectangularRoom представлено нижче:

```
public class RectangularRoom
{
   private int x, y, width, height;
   public RectangularRoom(int x, int y, int width, int height)
   {  this.x = x;
      this.y = y;
      this.width = width;
      this.height = height;
   }
   // ... інші методи }
```

Клас RectangularRoom також надає методи для зручної роботи з геометричними властивостями кімнати. Наприклад, метод Center() повертає координати центра кімнати:

```
public Vector2Int Center() => new Vector2Int(x + width / 2, y + height / 2);
```

Метод RandomPoint() дозволяє отримати випадкову внутрішню координату в межах кімнати, що використовується, наприклад, для розміщення гравця, сходів або сутностей:

```
public Vector2Int RandomPoint() => new Vector2Int(Random.Range(x + 1, x + width - 1), Random.Range(y + 1, y + height — 1));
```

Важливою функцією класу RectangularRoom є метод Overlaps(), який перевіряє, чи дана кімната перетинається з будь-якою іншою кімнатою зі списку. Ця перевірка виконується за допомогою Bounds:

```
public bool Overlaps(List<RectangularRoom> otherRooms)
{ foreach (RectangularRoom otherRoom in otherRooms)
   {  if (GetBounds().Intersects(otherRoom.GetBounds())) // Використання Bounds.Intersects для перевірки перекриття
      { return true; }
   }
```

```
    return false;
}
```

Алгоритм генерації підземелля у класі ProcGen використовує об'єкти RectangularRoom на наступних етапах:

Генерація та розміщення кімнат: Процес ітеративно намагається створити та розмістити задану максимальну кількість (maxRooms) прямокутних кімнат, представлених об'єктами RectangularRoom. Для кожної потенційної кімнати випадковим чином визначаються її розміри та позиція, після чого виконується перевірка на перекриття з вже успішно розміщеними кімнатами за допомогою методу Overlaps().

Вирізання кімнат на карті: Якщо нова кімната не перекривається, її форма "вирізається" на сітковій карті, заповнюючи відповідні клітинки стінами по периметру та підлогою всередині. Визначення периметру та внутрішньої області базується на координатах та розмірах об'єкта RectangularRoom.

З'єднання кімнат коридорами: Для всіх розміщених кімнат, окрім першої, прокладається коридор, що з'єднує її з попередньою кімнатою. Алгоритм TunnelBetween використовує координати центрів (Center()) двох об'єктів RectangularRoom для побудови шляху коридору. Координати для сегментів коридору обчислюються за алгоритмом лінії Брезенхема, який дозволяє визначити всі точки між двома заданими точками на сітці. Клітинки коридорів позначаються як підлога, а прилеглі порожні клітинки – як стіни, формуючи таким чином стіни коридору.

Розміщення сутностей та предметів: Після створення кімнат та коридорів у кожній кімнаті (окрім, як правило, стартової) випадковим чином розміщуються вороги та предмети (PlaceEntities). Розміщення відбувається у вільних внутрішніх клітинках кімнати, використовуючи, наприклад, метод RandomPoint().

Визначення стартової/фінішної позиції: Гравець розміщується у випадковій вільній клітинці першої кімнати (використовуючи RandomPoint()), а сходи на наступний рівень – у випадковій вільній клітинці останньої кімнати.

Таким чином, реалізований алгоритм процедурної генерації, що базується на роботі з об'єктами RectangularRoom, забезпечує динамічне створення унікальних лабіринтів, формуючи основу реіграбельності гри.

### 2.3.2. Опис імпортованих алгоритмів

Крім основного алгоритму процедурної генерації, у програмному засобі «Мандрівка Підземеллями» використано декілька стандартних алгоритмів, реалізації яких були адаптовані з зовнішніх джерел для потреб проєкту. Ці алгоритми формують частину алгоритмічної основи гри, забезпечуючи функціональність руху, сприйняття та допоміжних геометричних обчислень.

Алгоритм пошуку шляху A. Реалізований у класі AStar.cs. A* є ефективним алгоритмом пошуку найкоротшого шляху на графі або сітці з ваговими ребрами. У даному проєкті він застосовується для визначення оптимального шляху переміщення ворогів по сітці підземелля, враховуючи наявність перешкод.

Принцип роботи A* ґрунтується на оціночній функції f(n) = g(n) + h(n), де g(n) – фактична вартість шляху від старту до поточного вузла n, а h(n) – евристична оцінка відстані від n до цілі. Алгоритм ітеративно розглядає вузли карти, розширюючи шлях у напрямку вузла з найменшим f у відкритому списку, поки не досягне цілі. У реалізації використовується клас Node для представлення вузлів шляху з їхніми оцінками та посиланням на батьківський вузол. Вартість переміщення між сусідніми клітинками (gScore) враховує тип руху (ортогональний чи діагональний).

Фрагмент коду, що ілюструє основний цикл алгоритму A*:

```
public Vector2 Compute(Vector2Int start, Vector2Int goal) {
   // ... ініціалізація openList, closedList
   while (openList.Count > 0 && path == null) {
      // ... пошук сусідів, обчислення оцінок
      UpdateCurrentTile(ref currentNode); // Вибір наступного вузла
      path = GeneratePath(currentNode, start, goal); // Перевірка цілі
   }
```

```
    // ... повернення шляху
}
```

Алгоритм визначення поля зору (Field of View - FoV). Реалізований у класах AdamMilVisibility.cs та Visibility.cs. Цей алгоритм обчислює видиму область на сітковій карті з певної точки, враховуючи перешкоди. У проєкті використано алгоритм, що базується на методі Адама Міла, відомого своєю ефективністю для сіткових ігор.

Алгоритм працює шляхом рекурсивного сканування секторів карти, виходячи з початкової точки (позиції гравця). Він використовує поняття "схилів" (клас Slope у реалізації) для відстеження меж видимості в кожному стовпці, віддаленому від початкової точки. При зустрічі з перешкодою (стіною) межі видимості коригуються, блокуючи подальший огляд у цьому напрямку.

Фрагмент коду, що показує структуру AdamMilVisibility:

```
sealed class AdamMilVisibility : Visibility {
    // ... поля та внутрішні структури (Slope)
    public override void Compute(Vector3Int origin, int rangeLimit, List<Vector3Int> fieldOfView) {
        fieldOfView.Add(origin); // Початкова точка завжди видима
        for (uint octant = 0; octant < 8; octant++) {
            // Рекурсивний виклик для 8 октантів
            Compute(octant, origin, rangeLimit, 1, new Slope(1, 1), new Slope(0, 1), fieldOfView);
        }
    }// ... рекурсивний метод Compute з логікою сканування}
```

Результат роботи алгоритму – список видимих клітинок – використовується для оновлення відображення карти гравцеві, реалізуючи механіку "туману війни".

Алгоритм лінії Брезенхема. Реалізований у статичному класі BresenhamLine.cs. Цей класичний алгоритм використовується у проєкті для визначення всіх цілочисельних координат на сітці, що лежать на прямій лінії між

двома заданими точками. У контексті процедурної генерації підземелля він застосовується для точного прокладання коридорів між кімнатами.

Алгоритм працює ітеративно, визначаючи наступну клітинку на лінії шляхом аналізу помилки відхилення від ідеальної прямої, використовуючи лише цілочисельні операції.

Фрагмент коду, що показує основний цикл алгоритму Брезенхема:

```
static public class BresenhamLine
{
  static public void Compute(Vector2Int start, Vector2Int end, List<Vector2Int> coords)
  {
    // ... ініціалізація
    while (true)
    {
      coords.Add(new Vector2Int(x, y)); // Додаємо поточну клітинку
      if (x == end.x && y == end.y) { break; } // Умова зупинки
      // Логіка вибору наступної клітинки
      int e2 = 2 * err;
      if (e2 > -dy) { err -= dy; x += sx; }
      if (e2 < dx) { err += dx; y += sy; }}}}
```

Цей алгоритм є допоміжним інструментом, що забезпечує коректну побудову геометрії коридорів на дискретній сітці карти.

### 2.3.3. Огляд імпортованої бібліотеки OdinSerializer

Реалізація системи збереження та завантаження ігрового прогресу є критично важливою функціональністю для більшості сучасних ігор, включаючи проєкти в жанрі RogueLike, де гравці часто бажають продовжити проходження з останнього збереженого моменту. Процес збереження стану гри передбачає серіалізацію – перетворення структурованих даних об'єктів у формат, придатний для зберігання (наприклад, у файл) або передачі, а завантаження – відповідно,

десеріалізацію – відновлення об'єктів з цих даних. Враховуючи складність ігрового стану в розробленій RogueLike грі, що охоплює динамічно згенеровану карту, різноманітні сутності з їхніми унікальними характеристиками та станом, вміст інвентарів, взаємозв'язки між об'єктами та інші параметри ігрового світу, вибір ефективного та надійного інструменту для серіалізації набув особливого значення.

Для розв'язання цієї задачі у проєкті було використано імпортовану бібліотеку OdinSerializer. Ця бібліотека є потужним та гнучким інструментом серіалізації, спеціально розробленим для використання в середовищі Unity, який забезпечує високу продуктивність та надійність. Однією з ключових переваг OdinSerializer є його здатність коректно обробляти складні графіки об'єктів, включаючи циркулярні посилання та поліморфні типи, що часто зустрічається в ігровій логіці. Традиційні методи серіалізації в .NET або стандартні підходи в Unity можуть мати обмеження при роботі з такими структурами, що вимагає значних додаткових зусиль для їх обробки вручну. OdinSerializer автоматизує ці процеси, дозволяючи серіалізувати практично будь-які типи даних, що використовуються в Unity та C#.

Використання OdinSerializer дозволило уникнути необхідності розробки власної складної та потенційно схильної до помилок системи серіалізації з нуля. Бібліотека надає простий та інтуїтивно зрозумілий API для виконання операцій збереження та завантаження, інтегруючись безпосередньо з об'єктами Unity та C# класами, що представляють ігровий стан (як це реалізовано у класі SaveManager.cs). Це суттєво спростило реалізацію функціоналу збереження/завантаження, дозволивши зосередитися на інших аспектах розробки гри. Завдяки своїй ефективності та надійності, OdinSerializer виступив як оптимальний вибір для забезпечення можливості збереження ігрового прогресу в розробленій RogueLike грі, що є важливим елементом зручності та функціональності програмного засобу.

## 2.4. Обґрунтування вибору інструментальних засобів розробки програмного засобу «Мандрівка Підземеллями»

Процес розробки програмного засобу «Мандрівка Підземеллями» вимагав ретельного обґрунтування вибору технологічної платформи та інструментальних засобів, що були б оптимальними для реалізації всіх запланованих функціональних та нефункціональних вимог в рамках встановлених термінів бакалаврської роботи. Вибір інструментарію визначався необхідністю ефективної реалізації специфічних для жанру RogueLike механік, забезпеченням гнучкості розробки, можливостями кросплатформенного розгортання та доступністю якісної документації і підтримки. На основі проведеного аналізу та з урахуванням особливостей проєкту, основним інструментальним засобом було обрано ігровий рушій Unity у поєднанні з мовою програмування C#, доповнені специфічними бібліотеками.

Вибір ігрового рушія Unity як фундаментальної основи проєкту обумовлений його високою універсальністю та адаптивністю до завдань різної складності, що робить його одним із лідерів серед інді-розробників. Для реалізації RogueLike гри, яка зазвичай використовує двовимірну графіку та сіткову структуру світу, важливим аспектом стала потужна підтримка 2D розробки в Unity, включаючи спеціалізовані інструменти для роботи зі спрайтами, анімацією, 2D фізикою та системою тайлових карт, що значно спрощує створення ландшафту підземелля та розміщення на ньому об'єктів. Гнучка компонентно-орієнтована архітектура рушія ідеально відповідає дизайну ігрових сутностей, дозволяючи легко додавати, модифікувати та комбінувати функціональність через компоненти, що є основою модульності та розширюваності проєкту. Візуальний редактор Unity з його інтуїтивно зрозумілим інтерфейсом та можливістю роботи зі сценами, префабами та налаштуванням об'єктів у реальному часі суттєво прискорив процес прототипування та налагодження ігрових механік. Крім того, широка кросплатформенність Unity надає потенційну можливість легко адаптувати гру

для запуску на різних операційних системах та пристроях, що є важливою перспективою для будь-якого ігрового проєкту. Доступ до великої екосистеми через Unity Asset Store та активна спільнота розробників стали додатковими перевагами, надаючи доступ до готових ресурсів та рішень, а також можливість отримати підтримку при виникненні проблем.

Мова програмування C# була обрана як основний інструмент для написання всієї ігрової логіки та алгоритмів, що повністю відповідає стандартам розробки в середовищі Unity. C# є сучасною, сильно типізованою та об'єктно-орієнтованою мовою, яка забезпечує високу продуктивність виконання скриптів у рушії . Її синтаксис є зрозумілим та виразним, що сприяло ефективній реалізації складних алгоритмів, таких як процедурна генерація підземелля, пошук шляху A* та обчислення поля зору. Об'єктно-орієнтовані можливості C# добре узгоджуються з компонентною архітектурою Unity та принципами проектування системи сутностей та їхньої взаємодії. Наявність розвиненої бібліотеки класів .NET також розширила можливості розробки, надаючи готові рішення для стандартних задач. Таким чином, поєднання Unity та C# створило потужне та гнучке середовище, оптимальне для реалізації всіх технічних аспектів RogueLike гри.

Окрім основних інструментів, для вирішення специфічних задач були використані додаткові бібліотеки. Інструмент TextMeshPro був інтегрований для забезпечення високоякісного та ефективного відображення тексту в елементах користувацького інтерфейсу. Це було особливо важливо для реалізації ігрового логу повідомлень та панелей статистики, де необхідно було динамічно оновлювати та форматувати текст. Вибір TextMeshPro зумовлений його розширеними можливостями форматування, кращою візуальною якістю тексту та оптимізацією рендерингу порівняно зі стандартними засобами Unity UI Text. Для реалізації системи збереження та завантаження стану гри було обрано бібліотеку OdinSerializer. Враховуючи складність ігрового стану в RogueLike (позиції сутностей, їхні характеристики, вміст інвентарів, стан карти, параметри генерації), необхідним був надійний та гнучкий інструмент серіалізації.

OdinSerializer вирізняється високою продуктивністю та здатністю коректно серіалізувати складні структури об'єктів та посилання між ними, що є критично важливим для коректного збереження та відновлення прогресу гри. Використання цієї бібліотеки дозволило уникнути необхідності реалізації власної складної системи серіалізації.

Таким чином, ретельно обраний комплекс інструментальних засобів, що включає ігровий рушій Unity, мову програмування C# та додаткові бібліотеки TextMeshPro і OdinSerializer, є оптимальним для розробки програмного засобу «Мандрівка Підземеллями» в рамках бакалаврської роботи. Цей набір інструментів забезпечив всі необхідні можливості для ефективної реалізації запланованого функціоналу, створення гнучкої архітектури, оптимізації окремих аспектів розробки та досягнення якісного кінцевого результату, відповідного вимогам жанру RogueLike.

## 2.5. Особливості програмної реалізації та основні режими роботи програмного засобу «Мандрівка Підземеллями»

Реалізація програмного засобу «Мандрівка Підземеллями» здійснювалась у середовищі розробки Unity з використанням мови програмування C#, що дозволило ефективно втілити концепцію RogueLike гри. Весь процес програмної реалізації включав розробку та інтеграцію ключових ігрових систем та компонентів, кожен з яких відповідав за певний аспект функціонування гри. Особлива увага приділялася реалізації процедурної генерації, покрокової механіки та взаємодії сутностей.

Реалізація базових сутностей та компонентів. Основою ігрового світу є система сутностей, яка реалізована за допомогою успадкування та компонентного підходу Unity. Базовий клас Entity (Entity.cs) інкапсулює спільні властивості всіх ігрових об'єктів на карті (позиція, візуальне представлення, базові дії). Нащадки Entity, такі як клас Player (Player.cs), розширюють його функціонал специфічною логікою. Ключові характеристики та поведінка

сутностей визначаються прикріпленими до них компонентами. Наприклад, бойові атрибути та логіка бою реалізовані у компоненті Fighter.cs, а система інвентарю – у компоненті Inventory.cs.

Реалізація управління картою та процедурної генерації. Ігровий світ, підземелля, генерується процедурно при кожному новому проходженні. За це відповідає клас ProcGen.cs, який використовує алгоритм генерації на основі розміщення кімнат, з'єднаних коридорами. Структуру окремої кімнати описує клас RectangularRoom, який містить методи для перевірки перекриття та отримання координат. Клас MapManager.cs використовує результати роботи ProcGen для управління ігровою картою, яка представлена набором тайлів (TileData.cs). Візуалізація карти здійснюється, ймовірно, за допомогою системи Tilemap Unity.

Реалізація штучного інтелекту супротивників. Поведінка ворогів визначається реалізацією їхнього штучного інтелекту. Класи, що успадковують від базового класу AI.cs, містять логіку прийняття рішень для різних типів супротивників. Для навігації ворогів по підземеллю використовується алгоритм пошуку шляху A* (AStar.cs), який розраховує оптимальний маршрут до цілі, враховуючи перешкоди. Визначення того, чи бачить ворог гравця, відбувається за допомогою алгоритму поля зору (AdamMilVisibility.cs).

Реалізація системи інвентарю та предметів. Система інвентарю реалізована як компонент Inventory.cs, що дозволяє сутностям взаємодіяти з предметами. Предмети представлені класами, що успадковують від Item.cs та Consumable.cs (для споживаних). Кожен тип предмета (наприклад, Healing.cs, Fireball.cs) реалізує свій унікальний ефект у методі Use().

Реалізація інтерфейсу користувача. Інтерфейс користувача (UI) розроблено за допомогою системи Unity UI. Клас UIManager.cs відповідає за відображення всіх елементів UI на екрані: ігрового поля, панелей статистики гравця (здоров'я, інвентар), логу повідомлень. Оновлення інтерфейсу відбувається динамічно в залежності від стану гри та дій гравця. Для відображення тексту використовується TextMeshPro.

Реалізація управління грою та режимів роботи. Загальний потік гри та управління режимами роботи здійснюється класом GameManager.cs. Він контролює перехід між основними станами гри:

- головне меню (MainMenu.cs): Початковий режим, де можна розпочати нову гру, завантажити збережену (з використанням SaveManager.cs) або вийти;
- ігровий процес: Основний режим взаємодії з підземеллям. В цьому режимі обробляється введення гравця, відбуваються ходи сутностей, оновлюється стан світу та UI;
- екран завершення гри: Відображається при смерті гравця або виконанні іншої умови завершення. Показує результат проходження;
- пауза: Режим, що призупиняє гру.

Реалізація системи збереження/завантаження. Збереження та завантаження прогресу гри реалізовано у класі SaveManager.cs з використанням бібліотеки OdinSerializer для ефективної серіалізації/десеріалізації складних даних проєкту.

Програмна реалізація всіх зазначених систем та компонентів, їхня взаємодія та інтеграція формують функціональний прототип RogueLike гри.

## 2.6. Організація тестування та налагодження програмного засобу «Мандрівка Підземеллями»

Тестування та налагодження становили невід'ємну частину процесу розробки програмного засобу "RogueLike гра". Ці етапи були інтегровані в кожну ітерацію розробки, що відповідає принципам обраної ітеративної моделі. Основний підхід до тестування полягав у проведенні ручних перевірок реалізованої функціональності та ігрових механік безпосередньо в середовищі розробки Unity.

На кожному етапі впровадження або модифікації окремих модулів проводилося детальне тестування для забезпечення їхньої коректної роботи. Зокрема, здійснювалася перевірка функціональності процедурної генерації підземель, оцінювалася варіативність та правильність структури згенерованих

рівнів. Також ретельно тестувалася покрокова механіка руху та взаємодії всіх ігрових сутностей на сітці. Важливою частиною тестування була перевірка бойової системи, що включала розрахунок та застосування шкоди, а також коректне відображення стану здоров'я персонажів та умов завершення бою. Функціональність системи інвентарю та предметів також підлягала тестуванню, перевірялася можливість підбору, зберігання, використання та відкидання предметів, а також правильність застосування їхніх унікальних ефектів на персонажа гравця. Спостерігалася та аналізувалася поведінка супротивників, тестувалася їхня здатність до навігації за допомогою алгоритму пошуку шляху та коректність роботи системи визначення поля зору. Окрім ігрових механік, тестувався користувацький інтерфейс на предмет правильного відображення ігрової інформації та функціональності елементів меню. На завершальних етапах перевірялася працездатність системи збереження та завантаження ігрового прогресу.

Налагодження програмного засобу здійснювалося паралельно з тестуванням з використанням вбудованих інструментів середовища розробки Unity. Для відстеження проміжних значень, послідовності виконання коду та ідентифікації місць виникнення помилок активно використовувався консольний вивід (Debug.Log). У більш складних випадках застосовувався вбудований відлагоджувач Unity, що дозволяв покроково аналізувати виконання коду, встановлювати точки зупину та інспектувати стан програми. Тестування та налагодження в режимі Play в Unity Editor давало змогу оперативно виявляти та виправляти дефекти в процесі ігрової симуляції. Застосування цих методів налагодження дозволило ідентифікувати причини виявлених помилок та ефективно їх усувати, що сприяло підвищенню стабільності роботи прототипу. В результаті проведення тестування та налагодження було виправлено значну частину програмних недоліків, що забезпечило працездатність основного функціоналу гри в межах, передбачених завданнями бакалаврської роботи. Слід зазначити, що через обмежені терміни проєкту повне та вичерпне тестування всіх можливих сценаріїв взаємодії не проводилось.

## 2.7. Рекомендації по використанню та впровадженню програмного засобу

Розроблений в рамках даної бакалаврської роботи програмний засіб «Мандрівка Підземеллями», який являє собою функціональний прототип RogueLike гри на Unity, демонструє реалізацію ключових механік жанру та може знайти своє практичне застосування у декількох напрямках. Насамперед, він може виступати як самостійний невеликий ігровий продукт, що надає користувачеві базовий інтерактивний досвід дослідження процедурно згенерованого підземелля. Зважаючи на високу реіграбельність, притаманну жанру RogueLike, навіть прототип з обмеженим контентом може забезпечити певний рівень зацікавленості.

По-друге, програмний засіб може слугувати основою для подальшої розробки повноцінної комерційної або безкоштовної гри. Реалізовані ключові системи – процедурна генерація, покроковий бій, інвентар, базовий AI – є міцним фундаментом, на який можна нарощувати новий контент (нові типи ворогів, предметів, механік, рівнів), вдосконалювати графіку та звук, розширювати сюжетні елементи та оптимізувати продуктивність.

По-третє, проєкт може бути використаний як навчальний матеріал або портфоліо. Вивчення його структури та коду може допомогти іншим початківцям в освоєнні розробки ігор на Unity, зокрема реалізації складних алгоритмів процедурної генерації, покрокових систем та архітектури ігрових проєктів з використанням компонентного підходу та C#. Для автора роботи, розроблений програмний засіб є значущим елементом портфоліо, що демонструє практичні навички у сфері геймдеву.

Для забезпечення коректної роботи програмного засобу «Мандрівка Підземеллями» необхідно дотриматися певних умов експлуатації та наявності необхідного технічного обладнання. Оскільки гра розроблена на Unity як десктопний додаток, вона призначена для запуску на персональних комп'ютерах під управлінням відповідних операційних систем, що підтримуються Unity (наприклад, Windows, macOS, Linux).

# ВИСНОВКИ

Під час виконання бакалаврської роботи було проведено комплексне дослідження та практичну реалізацію програмного засобу в жанрі RogueLike.

Було опрацьовано теоретичні відомості, що стосуються історії, ключових особливостей та еволюції жанру RogueLike, а також проведено аналіз популярних ігор цього жанру, що дозволило виявити успішні підходи та сформувати вимоги до власної розробки.

Досліджено сучасні технології та інструментальні засоби розробки інді-ігор, зокрема ігровий рушій Unity та мову програмування C#, обґрунтовано їх вибір для реалізації поставленої задачі, враховуючи специфіку проєкту та можливості інструментарію.

Розроблено концепцію та спроектовано архітектуру програмного засобу «Мандрівка Підземеллями», яка базується на компонентно-орієнтованому підході Unity та об'єктно-орієнтованому програмуванні, з урахуванням вимог до функціоналу та структури ігрового світу. Побудовано діаграму класів, що ілюструє основні сутності та їхні взаємозв'язки.

Реалізовано функціональний прототип RogueLike гри на базі Unity. В ході програмної реалізації розроблено та інтегровано ключові ігрові механіки, зокрема:

- систему процедурної генерації підземелля на основі розміщення кімнат та з'єднання їх коридорами;
- покрокову систему руху та взаємодії сутностей;
- бойову систему з розрахунком шкоди та станом сутностей;
- систему інвентарю та використання предметів з унікальними ефектами;
- базовий штучний інтелект для супротивників з використанням алгоритму пошуку шляху A* та визначення поля зору;
- користувацький інтерфейс для відображення ігрової інформації та взаємодії;
- систему збереження та завантаження прогресу гри.

Проведено тестування та налагодження програмного засобу, в ході якого виявлено та усунено значну частину програмних помилок, забезпечено стабільну роботу реалізованого функціоналу в основних режимах гри.

Створений програмний засіб відповідає поставленим у роботі задачам та демонструє працездатність реалізованих механік, підтверджуючи можливість створення комплексних ігор жанру RogueLike силами одного розробника з використанням сучасних інструментів.

Незважаючи на досягнуті результати, у прототипі присутні певні обмеження та недоліки, що обумовлені обсягом роботи та часовими рамками.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Berlin interpretation. RogueBasin. URL: https://www.roguebasin.com/index.php/Berlin_Interpretation (date of access: 04.05.2025).
2. Brazie A. Best game engines for beginner game developers in 2024. Game Design Skills. URL: https://gamedesignskills.com/game-development/video-game-engines/ (date of access: 30.04.2025).
3. Brogue. Official brogue site. URL: https://sites.google.com/site/broguegame/ (date of access: 04.05.2025).
4. Enter the gungeon. Wikipedia. URL: https://uk.wikipedia.org/wiki/Enter_the_Gungeon (date of access: 04.05.2025).
5. Gaming engines. Arm. URL: https://www.arm.com/glossary/gaming-engines (date of access: 30.04.2025).
6. Genre, prototype theory and the berlin interpretation of roguelikes. GameStudies. URL: https://gamestudies.org/2403/articles/cartlidge (date of access: 04.05.2025).
7. Hades. Wikipedia. URL: https://uk.wikipedia.org/wiki/Hades (date of access: 04.05.2025).
8. Rogue. RogueBasin. URL: https://www.roguebasin.com/index.php/Rogue (date of access: 04.05.2025).
9. The binding of isaac. Wikipedia. URL: https://uk.wikipedia.org/wiki/The_Binding_of_Isaac (date of access: 04.05.2025).
10. Unity manual. Unity Technologies. URL: https://docs.unity3d.com/Manual/ (date of access: 30.04.2025).
11. Unity. Official unity site. URL: https://unity.com/ (date of access: 30.04.2025).
12. What is core to a Roguelike?. Vigaroe. URL: https://www.vigaroe.com/2019/08/what-is-core-to-roguelike.html (date of access: 04.05.2025).

# ДОДАТОК А
# ТЕХНІЧНЕ ЗАВДАННЯ НА РОЗРОБКУ ROGUELIKE ГРИ З ДОПОМОГОЮ UNITY

## ВСТУП

Дане технічне завдання визначає вимоги до програмного забезпечення «Мандріка Підземеллями», яке розробляється в рамках бакалаврської роботи. Гра належить до жанру RogueLike, що характеризується процедурною генерацією рівнів та перманентною смертю.

## ПІДСТАВИ ДЛЯ РОЗРОБКИ

Розробка «Мандріка Підземеллями» на Unity здійснюється на підставі написання бакалаврської роботи.

## ПРИЗНАЧЕННЯ РОЗРОБКИ

Програмний засіб призначений для:

- надання користувачеві інтерактивного досвіду дослідження процедурно згенерованих підземель;
- боротьби з монстрами та пошуку предметів в умовах покрокового ігрового процесу;
- демонстрації можливостей ігрового рушія Unity для реалізації комплексних ігрових механік жанру RogueLike;
- слугування основою для подальшого розвитку проєкту;
- використання як навчального матеріалу або елементу портфоліо.

## ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

**Вимоги до функціональних характеристик**: Програмний засіб повинен бути інтуїтивно зрозумілим для користувача. В ньому мають бути реалізовані такі функції:

- система процедурної генерації підземель, яка створює унікальну структуру рівня при кожному новому проходженні;
- покрокова система руху та взаємодії, де дії всіх сутностей відбуваються послідовно;

- бойова система, що дозволяє персонажам атакувати одне одного з розрахунком шкоди;
- система ігрових сутностей, що включає гравця та різні типи ворогів з визначеними характеристиками;
- базовий штучний інтелект (AI) для ворогів, що визначає їхню поведінку;
- система інвентарю, яка дозволяє гравцеві підбирати, зберігати та використовувати предмети;
- система предметів з різними типами інтерактивних об'єктів, що мають унікальні властивості;
- система "туману війни" (Fog of War) та поля зору (Field of View);
- коректна обробка переходу між рівнями підземелля;
- умова завершення гри (наприклад, смерть гравця);
- система збереження та завантаження поточного прогресу гри;
- візуальне представлення ігрового світу (ігрове поле, персонажі, об'єкти);
- інтерфейс користувача з відображенням стану гравця (здоров'я, рівень тощо);
- візуальний доступ до інвентарю;
- лог повідомлень про значущі події під час гри;
- головне меню з опціями нової гри, завантаження та виходу.

**Вимоги до надійності:** система повинна працювати без збоїв та затримок на персональних комп'ютерах.

**Умови експлуатації:** програмний засіб може використовуватися на персональних комп’ютерах та ноутбуках з операційними системами, що підтримуються Unity.

Вимоги до інформаційної і програмної сумісності: Гра повинна працювати на операційних системах Windows, Mac OS, Linux. Розроблена на базі Unity з використанням C#. Використано бібліотеки TextMeshPro та OdinSerializer.

## ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу документів входять бакалаврська робота та технічне завдання.

## ТЕХНІКО-ЕКОНОМІЧНІ ПОКАЗНИКИ

Після доопрацювання застосунок може бути розміщений у вільному доступі або розглянутий для комерційного розповсюдження. Економічну ефективність можна буде оцінити після виходу на широке коло користувачів.

## СТАДІЇ І ЕТАПИ РОЗРОБКИ

Розробка здійснюється за ітеративною моделлю. Етапи включають:

1. Формування та аналіз вимог.
2. Проектування архітектури та модулів.
3. Реалізація (кодування) з використанням Unity та C#.
4. Тестування та налагодження реалізованого функціоналу.
5. Впровадження системи (збірка та розгортання)
6. Супровід (подальший розвиток). Кожна ітерація включає міні-етапи аналізу, проектування, реалізації та тестування

## ПОРЯДОК КОНТРОЛЮ І ПРИЙМАННЯ

Приймання результатів роботи здійснюється в процесі здачі бакалаврської роботи.

# ДОДАТОК Б
# ІНСТРУКЦІЯ КОРИСТУВАЧУ ПРОГРАМНОГО ЗАСОБУ «МАНДРІВКА ПІДЗЕМЕЛЛЯМИ»

Комп'ютерна гра «Мандріка Підземеллями» є програмним засобом, розробленим з використанням ігрового рушія Unity. Гра реалізує ключові принципи жанру RogueLike, пропонуючи гравцеві дослідження процедурно згенерованих підземель та покроковий ігровий процес.

**Функціональне призначення**

Комп'ютерна гра «Мандріка Підземеллями» дозволяє користувачеві:

- досліджувати унікальні підземелля, що генеруються при кожному новому проходженні;
- взаємодіяти з ігровим оточенням, монстрами та предметами в покроковому режимі;
- брати участь у покрокових боях;
- збирати та використовувати різноманітні предмети з унікальними ефектами;
- спостерігати за станом свого персонажа та подіями у ігровому світі через інтерфейс користувача та лог повідомлень.

**Умови застосування програми**

Використання комп'ютерної гри доступне для пристроїв з операційною системою Windows, Mac OS, Linux. Для запуску необхідне відповідне технічне забезпечення персонального комп'ютера, що відповідає мінімальним вимогам рушія Unity для запуску зібраних проєктів.

**Опис роботи програми**

Початок гри відбувається після запуску виконуваного файлу застосунка. Гравець потрапляє до головного меню, де може обрати опції:

- нова гра: Розпочинає нове проходження зі випадково згенерованим підземеллям та стартовим персонажем;

- завантажити гру: Дозволяє відновити раніше збережений ігровий процес;

- вихід: Завершує роботу програми.

Під час ігрового процесу гравець керує персонажем на сітці підземелля. Рух та дії відбуваються покроково: гравець виконує дію, після чого всі інші сутності (вороги) виконують свої дії.

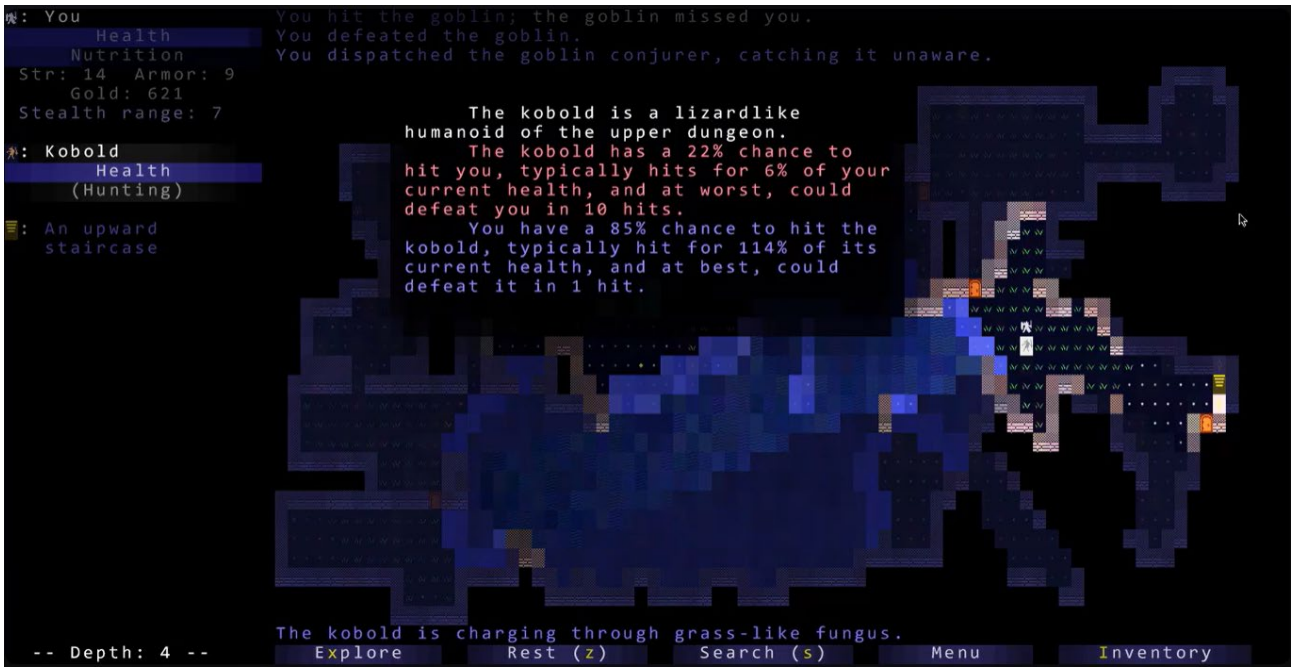
Керування здійснюється за допомогою клавіатури та миші.

На екрані відображається ігрове поле з підземеллям, персонажами та об'єктами, панелі зі статистикою гравця (здоров'я, рівень, інвентар), а також лог повідомлень, що інформує про події.

Суть гри полягає у дослідженні підземелля, боротьбі з ворогами, пошуку предметів та просуванні на наступні рівні підземелля через сходи. Гра завершується у разі смерті персонажа (перманентна смерть) або досягнення певного прогресу (якщо така умова реалізована).

Для завершення поточної гри та виходу з програми можна скористатися опцією "Вихід" у головному меню або, зазвичай, стандартною комбінацією клавіш операційної системи (наприклад, Alt + F4 на Windows ).

# ДОДАТОК В
# РИСУНКИ ПРОАНАЛІЗОВАНИХ ІГОР




Вигляд ігрового процесу у самій гри Brogue


Вигляд ігрового процесу у самій гри The Binding of Isaac

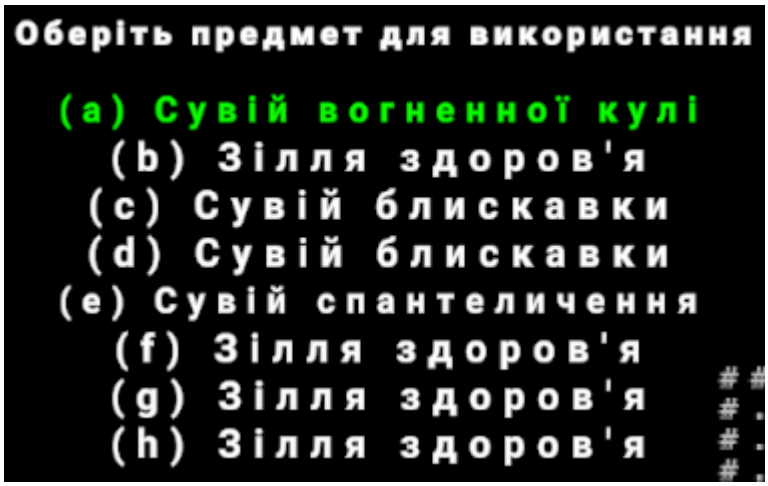

Вигляд ігрового процесу у самій гри Hades


Вигляд ігрового процесу у самій гри Enter the Gungeon

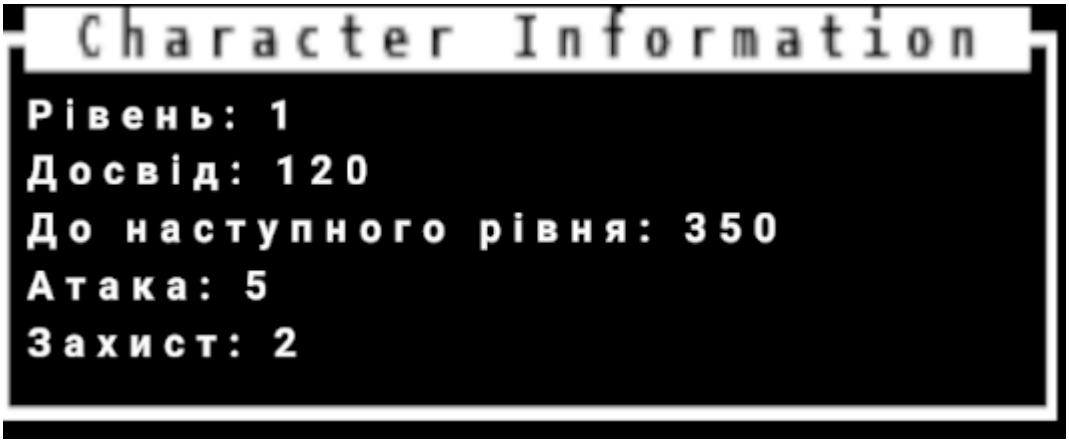
# ДОДАТОК Г
# РИСУНКИ З РОЗРОБЛЕНОЇ ГРИ «МАНДРІВКА ПІДЗЕМЕЛЛЯМИ»


Вигляд головного меню гри

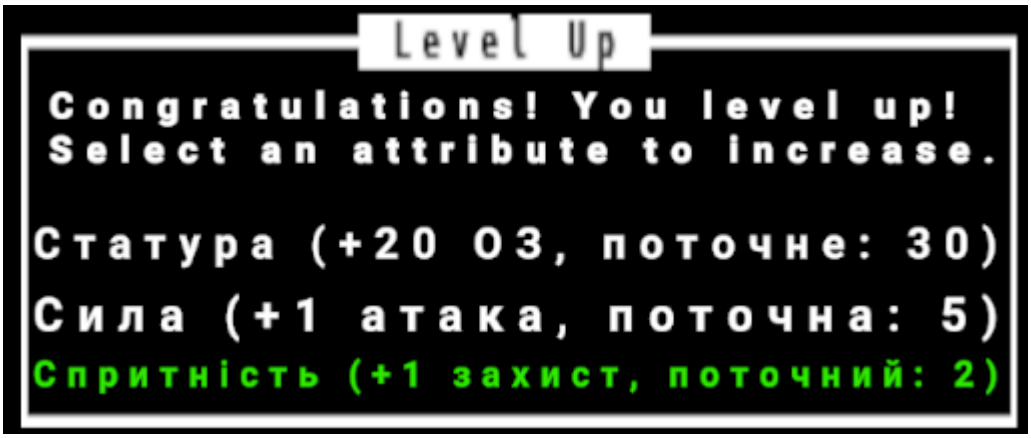

Вигляд ігрового процесу у самій грі



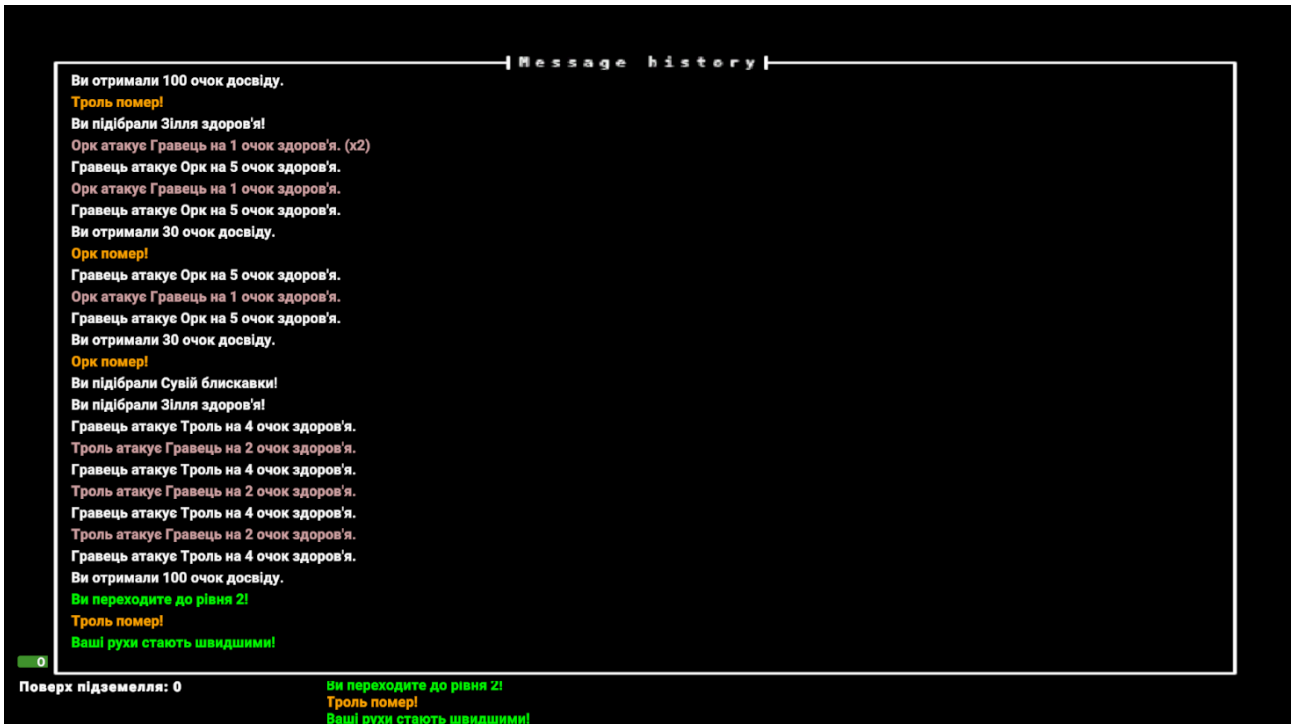

Вигляд відкритої вкладки «Інвентарь» з деякими предметами




Вигляд відкритої вкладки «Інформація про персонажа»




Вигляд сповіщення про підвищення рівня гравця




Вигляд відкритої вкладки «Історія повідомлень»

# АНОТАЦІЯ

**Кириченко В. С. Аналіз методів та засобів розробки комп'ютерних ігор жанру RogueLike на прикладі «Мандрівка Підземеллями». Рукопис.**

Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю 122 Комп’ютерні науки. Волинський національний університет імені Лесі Українки, Луцьк, 2025 р.

У цій кваліфікаційній роботі досліджено процес розробки функціонального прототипу RogueLike гри на Unity. Проведено аналіз жанру RogueLike, його ключових елементів та еволюції. Обґрунтовано вибір Unity та C# як основних інструментів розробки. Розглянуто теоретичні аспекти алгоритмів процедурної генерації та штучного інтелекту, що застосовуються в іграх цього жанру

Спроектовано архітектуру програмного засобу та реалізовано ключові ігрові механіки: процедурну генерацію рівнів, покрокову бойову систему, систему інвентарю, базовий штучний інтелект супротивників, користувацький інтерфейс та систему збереження/завантаження прогресу. Проведено тестування та налагодження розробленого прототипу. В результаті отримано працездатний прототип RogueLike гри, що демонструє можливості створення комплексних проєктів цього жанру на Unity.,

Ключові слова: RogueLike, Unity, C#, розробка ігор, процедурна генерація, штучний інтелект, ігрові механіки.

## ABSTRACT

**Kyrychenko V. S. Analysis of Methods and Tools for Developing RogueLike Computer Games, Using "Dungeon Journey" as an Example. Manuscript.**

Qualification Paper for the educational degree "Bachelor" in specialty 122 Computer Science. Lesya Ukrainka Volyn National University, Lutsk, 2025.

This qualification paper investigates the development process of a functional RogueLike game prototype using the Unity game engine. The work analyzes the RogueLike genre, its key elements, and evolution. The selection of Unity and C# as primary development tools was justified. Theoretical aspects of procedural generation algorithms and artificial intelligence applicable in this game genre were also explored.

The software's architecture was designed, and key game mechanics were implemented: procedural level generation, a turn-based combat system, an inventory system, basic enemy artificial intelligence, a user interface, and a save/load system. The developed prototype underwent testing and debugging to ensure its correct operation. As a result, a functional RogueLike game prototype was created, demonstrating Unity's capabilities for developing complex projects in this genre.



Keywords: RogueLike, Unity, C#, game development, procedural generation, artificial intelligence, game mechanics.