

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ ГРИ З ДИНАМІЧНОЮ
ГЕНЕРАЦІЄЮ МАПИ

SOFTWARE IMPLEMENTATION OF A GAME WITH
DYNAMIC MAP GENERATION

Виконав(ла): студент(ка) 4 курсу, групи КНТ-111
Спеціальності 121 Інженерія програмного
(код і найменування спеціальності)
забезпечення

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

БАКЛИЦЬКА А.Р.

(ПРИЗВИЩЕ та ініціали)

Керівник ПАРХОМЕНКО А.В.

(ПРИЗВИЩЕ та ініціали)

Рецензент ЗЕЛЕНЬОВА І.Я.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

БАКЛИЦЬКОЇ Анастасії Романівни

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація гри з динамічною генерацією мапи. Software Implementation of a Game with Dynamic Map Generation

керівник проєкту (роботи) к.т.н., доцент, ПАРХОМЕНКО Анжеліка Володимирівна,

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “28” квітня 2025 року № 209

2. Строк подання студентом проєкту (роботи) 02 червня 2025 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Дослідження алгоритмів генерації ігрової мапи. 3. Проєктування програмного забезпечення. 4. Конструювання програмного забезпечення. 5. Інструкція з використання програмного продукту

5. Перелік графічного матеріалу (з з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ПАРХОМЕНКО А.В., доцент		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання «14» квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділи 1, 2
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 3
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	ПЗ, Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Анастасія БАКЛИЦЬКА
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Анжеліка ПАРХОМЕНКО
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 86 с., 6 табл., 16 рис., 3 дод., 35 джерел.

КОМП'ЮТЕРНА ГРА, МАПА, АЛГОРИТМ ГЕНЕРАЦІЇ ЛАБІРИНТА, ІГРОВИЙ РУШІЙ.

Об'єкт дослідження – процес розробки програмного забезпечення для комп'ютерної гри.

Предмет дослідження – технології та алгоритми процедурної генерації ігрового середовища 3D-гри.

Мета роботи – програмна реалізація алгоритму побудови лабіринту для гри з механікою динамічної генерації мапи.

Матеріали, методи та технічні засоби: мова програмування C#, ігровий рушій Unity, алгоритм генерації лабіринту Recursive Backtracker.

Результати. Розроблено комп'ютерну 3D-гру з динамічною генерацією мапи та пошуком кінцевої точки, збором об'єктів та відображенням користувацького інтерфейсу.

Висновки. У процесі роботи було досліджено особливості розробки комп'ютерних ігор з процедурною генерацією мапи, реалізовано повноцінний ігровий цикл.

Галузь використання – комп'ютерні ігри.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 86 pages, 6 tables, 16 figures, 3 appendixes, 35 sources.

COMPUTER GAME, MAP, MAZE GENERATION ALGORITHM, GAME ENGINE.

The object of research – process of developing software for a computer game.

Subject of research – technologies and algorithms for procedural generation of the game environment of a 3D game.

The purpose of this work is software implementation of an algorithm for constructing a maze for a game with dynamic map generation mechanics.

Materials, methods and techniques: C# programming language, Unity engine, Recursive Backtracker maze generation algorithm.

Results. The 3D computer game with dynamic map generation and endpoint search, object collection and user interface display is developed.

Conclusions. In the course of the work, the peculiarities of developing games with procedural map generation were investigated, and a full-fledged game cycle was implemented.

The field of application – computer games.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ.....	9
1 Аналіз предметної області.....	10
1.1 Комп'ютерні ігри у сучасному світі.....	10
1.2 Особливості розробки ігор	11
1.3 Класифікація та аналіз комп'ютерних ігор	14
1.4 Аналоги комп'ютерних ігор з генерацією мапи	19
1.4.1 Spelunky.....	19
1.4.2 Deep Rock Galactic.....	20
1.4.3 Enter the Gungeon.....	21
1.4.4 Invisible, Inc.....	22
1.5 Постановка завдань дипломної роботи	24
2 Дослідження алгоритмів генерації ігрової мапи.....	25
2.1 Огляд та класифікація лабіринтів.....	25
2.2 Дослідження існуючих методів генерації лабіринтів	28
2.3 Висновки за розділом 2	38
3 Проєктування програмного забезпечення	39
3.1 Аналіз вимог до системи	39
3.2 Розробка архітектури системи	39
3.3 Порівняння ігрових рушіїв.....	40
3.3.1 Unity.....	40
3.3.2 Unreal Engine.....	41
3.3.3 Gotot.....	42
3.4 Вибір мови програмування	43
3.5 Вибір середовища програмування.....	44
3.5.1 Microsoft Visual Studio	44

3.5.2 JetBrains Rider	45
3.6 Висновки за розділом 3	46
4 Конструювання програмного забезпечення	47
4.1 Розробка ігрового процесу	47
4.1.1 Дія	47
4.1.2 Реакція	47
4.1.3 Відгук	48
4.2 Структура проєкту	48
4.3 Опис особливостей програмної реалізації.....	50
4.4 Опис реалізації графічного інтерфейсу	51
4.5 Висновки за розділом 4	52
5 Інструкція з використання програмного продукту.....	53
5.1 Системні вимоги.....	53
5.2 Функціонал системи	53
5.3 Інструкція з запуску системи.....	53
5.4 Опис роботи програми.....	54
5.5 Висновки до розділу 4	57
Висновки	58
Перелік джерел посилань	60
Додаток А Технічне завдання	63
Додаток Б Текст програми	67
Додаток В Слайди презентації.....	79

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

ARRG	– Augmented Reality Role-Plaing Game;
CRPG	– Computer Role-Playing Game;
FPS	– First-Person Shooter;
GDD	– Game Design Document;
MMORPG	– Massively Multiplayer Online Role-Playing Game;
MOBA	– Multiplayer Online Battel Arena;
RPG	– Role-Playing Game;
RTS	– Real-Time Strategy;
TPS	– Third-Person Shooter;
TRP	– Tactical Role-Playing Game;
ПЗ	– програмне забезпечення;
ПК	– персональний комп’ютер.

ВСТУП

Відеоігри завжди йшли поруч з відпочинком та комунікацією. Консолі та персональні комп'ютери дають нам змогу грати будь-де та будь з ким, хоч з сусідом по кімнаті, хоч з людиною на іншій частині земної кулі [1].

Нині в світі набирають популярність перезавпуски та римейки в усіх сферах від одягу до кіно, тож світ відеоігор ця тенденція теж не оминула. Вінтажні ігри користуються великою популярністю вже від моменту релізу і є популярними не тільки серед гравців-старожилів, а й у нових користувачів. Багато компаній скористалися цією тенденцією та випусти римейки своїх бестселерів [2].

Водночас розвиток технологій надає змогу не лише відтворювати старі концепції, а й створювати нові механіки, які раніше були неможливі через апаратні обмеження. Однією з сучасних тенденцій є процедурна генерація контенту – технологія, що дозволяє створювати унікальні світи без втручання розробника. Вона дає змогу зробити ігровий процес більш варіативним та непередбачуваним.

Сучасні ігрові компанії дедалі частіше впроваджують динамічну генерацію рівнів, ворогів чи ресурсів. Що дозволяє значно економити ресурси не нехтуючи ігровим досвідом [3].

У цій роботі основна увага приділяється розробці гри з використанням алгоритмів процедурної генерації лабіринту, що дозволяє створити унікальні мапи про кожному новому запуску.

Метою роботи є програмна реалізація алгоритму побудови лабіринту для гри з механікою динамічної генерації мапи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Комп'ютерні ігри у сучасному світі

У сучасних реаліях ігри є невід'ємною частиною нашого життя. Це багатомільярдна індустрія, яка має масовий вплив. У світі налічується приблизно 3,32 мільярда активних гравців у відеоігри. Ця цифра зросла більш ніж на 1 мільярд всього за вісім років.

Доступно багато різних жанрів ігор від симуляторів реального життя до головоломок у фантастичних світах.

Окрім розваги існує й освітня роль. Деякі відеоігри призначені виключно для обміну знаннями та можуть допомогти учням вивчити широкий спектр предметів. Приймаючи інтерактивний формат вони знаходять великий інтерес в учнів.

Нещодавнє дослідження проведене на 2000 дітей у США показало, що ті, хто грає від трьох годин в день, показали кращі результати в тестах на когнітивні навички, включаючи контроль імпульсів і пам'ять [4].

Багато гравців у відеоігри вважають позитивним аспект спільноти ігрового світу. Дійсно, багатокористувацькі ігри можуть допомогти гравцям створити дружбу та приєднатися до груп і мереж, які виходять за межі географічних і культурних кордонів. Існує відчуття причетності, яке приваблює багатьох до ігрового світу, що допомагає гравцям розвивати певні соціальні навички, включаючи здатність до ефективного спілкування та співпраці.

Дослідники з Університетів Едж Хілл і Університету Йорка разом з Ковертом вивчали понад 700 гравців у багатокористувацькі онлайн-ігри (ММО) [4].

За словами дослідників, вибірка варіювалася від геймерів, які грали лише одну годину на тиждень, до тих, хто грав 30 і більше. Команда з'ясувала, як участь у ММО пов'язана з сильнішим почуттям соціальної ідентичності. Це

відобразилося на більших соціальних навичках, меншій самотності та вищій самооцінці серед респондентів [4].

1.2 Особливості розробки ігор

Розробка ігор – це процес, який охоплює як творчі так і технічні частини. Починаючи від ідеї і до релізу гра підлягає безлічі змін на шляху створення. А розробники втілюють у життя найкреативніші підходи [5].

Під час створення гри ми маємо пройти 7 основних етапів розробки, які представлено на рис. 1.1. Вони є загальними для всіх типів ігор [6].



Рисунок 1.1 – Етапи розробки гри [6]

Першим етапом є планування. Воно є одним із найважливіших етапів тому, що в ньому закладається основа та напрямок проєкту. Етап включає в себе дослідження та аналіз ринку, виявлення потенційних прогалин, які можна заповнити. Також сюди входить аналіз цільової аудиторії, який є одним з ключових факторів в успіху гри. Це дасть змогу визначити інтереси гравців та на їх основі побудувати найпривабливіший геймплей. Визначення жанру визначає основну структуру гри та фокусує її на певних ігрових елементах. Останньою частиною є планування ресурсів проєкту, які знадобляться для створення гри [5]-[7].

Наступним етапом є попереднє виробництво. Тут формується загальний концепт гри. Якись ідею стають ключовими, а від деяких доводиться відмовитися. На цьому етапі відбувається загальна підготовка: створення розкадровок, вибір головних механік та геймплею, нариси персонажів та аналіз сил команди. Все це лягає в основу детально прописаної документації. Результатом є створення міцної основи для початку розробки. Він також включає в себе кілька стадій [5]-[7]:

- створення документу дизайну гри (GDD);
- створення концепт-арту та прототипів;
- викладення технічних вимог;
- планування сюжетної лінії та персонажів;
- налагодження ігрової механіки;
- складання детального плану виробництва;
- налаштування засобів розробки та середовищ.

Виробництво. Основний етап роботи над проєктом, він включає в себе такі стадії:

- програмування. Цей етап є серцем виробництва. Розробники перетворюють створений документ на код реальної гри, забезпечуючи функціонал та логіку;
- графіка та дизайн. На цьому етапі створюється стилістика гри. Дизайнери реалізують персонажів, анімації, фони та об'єкти;
- дизайн рівнів. Створюються макети карт, розміщуються об'єкти та завдання. Їх завдання максимально залучити гравця;
- звукове оформлення. Аудіо-контент гри відіграє важливу роль у сприйнятті та залученості гравців, створюючи імерсивний досвід.

Також на цьому етапі проводиться перевірка всіх аспектів гри. Це потрібно для усунення всіх помилок, збоїв та блокувань, які могли трапитися [5]-[7].

Тестування. Це систематичний процес, де тестери дають свої оцінки грі, для виявлення помилок та інших проблем. Етап є ключовим для коректного функціонування гри. Тестування включає в себе такі частини:

- тестування та оптимізація продуктивності;
- тестування на сумісність;
- тестування на відповідність нормам;
- бета-тестування.

Цикл тестування повторюється до тих пір, доки не буде усунуто всі критичні помилки в роботі гри. Спираючись на свої ресурси команда має усунути максимально можливу кількість помилок перед релізом продукту [4]-[7].

Передзапуск. На цей момент в команди вже має бути стабільна бета-версія гри, яку можна представити. Тут включається в роботу маркетинг. Для представлення гри можна створити проморолик, демонстрацію чи випустити ранню версію гри. Також ефективним варіантом реклами може бути представлення демоверсії гри на фестивалях таких як, E3 і PAX , або презентаціями, як-от Nintendo Directs чи State Of Play [5]-[7].

Запуск. Знаменна подія для кожної гри кульмінації її шляху. Це момент запуску гри для громадськості, роблячи її доступною для завантажень чи покупок. Цей процес починається з розповсюдження гри на різні платформи.

Постпродакшин. Останній етап, який зосереджений на продовженні життя гри за допомогою оновлень, вдосконалень та постійної підтримки. Тут виправляються всі незначні помилки, які залишилися на момент випуску гри та нові баги знайдені користувачами. В деяких жанрах випускаються патчі балансу для кращої стабілізації гри. А також додається новий DLC для підтримки уваги та цікавості гравців [5]-[7].

1.3 Класифікація та аналіз комп'ютерних ігор

Класифікація ігор систематизує їх за ігровими стратегіями. Можна виділити три загальні групи: дії, контролю та інформаційні.

Інформаційні ігри спрямовані на проведення навчального процесу шляхом гейміфікації процесу засобами «проблемного» навчання.

Ігри дії є емуляторами тренувань. Вони спрямовані на покращення реакцій та точності рухів.

Ігри контролю передбачають створення власних стратегій для отримання переваги [8]-[9].

Раніше ігри можна було поділити на чітко окреслені жанри, проте сьогодні кількість жанрів та піджанрів зростає, а межі між ними стрімко стираються. Розробники змішують декілька жанрів для створення нових і несподіваних рішень, а в деяких випадках ігри взагалі не можна віднести до жодного з існуючих жанрів. Проте, наразі все одно можна виокремити деякі категорії (рис. 1.2):

- пісочниця;
- стратегія в реальному часі;
- шутер;
- багатокористувацька онлайн бойова арена;
- рольові ігри;
- симулятори та спорт;
- головоломки та ігри для компанії;
- бойовик-пригода;
- ігри на виживання та жахи;
- платформери;
- текстові пригодницькі та візуальні новели [8]-[9].

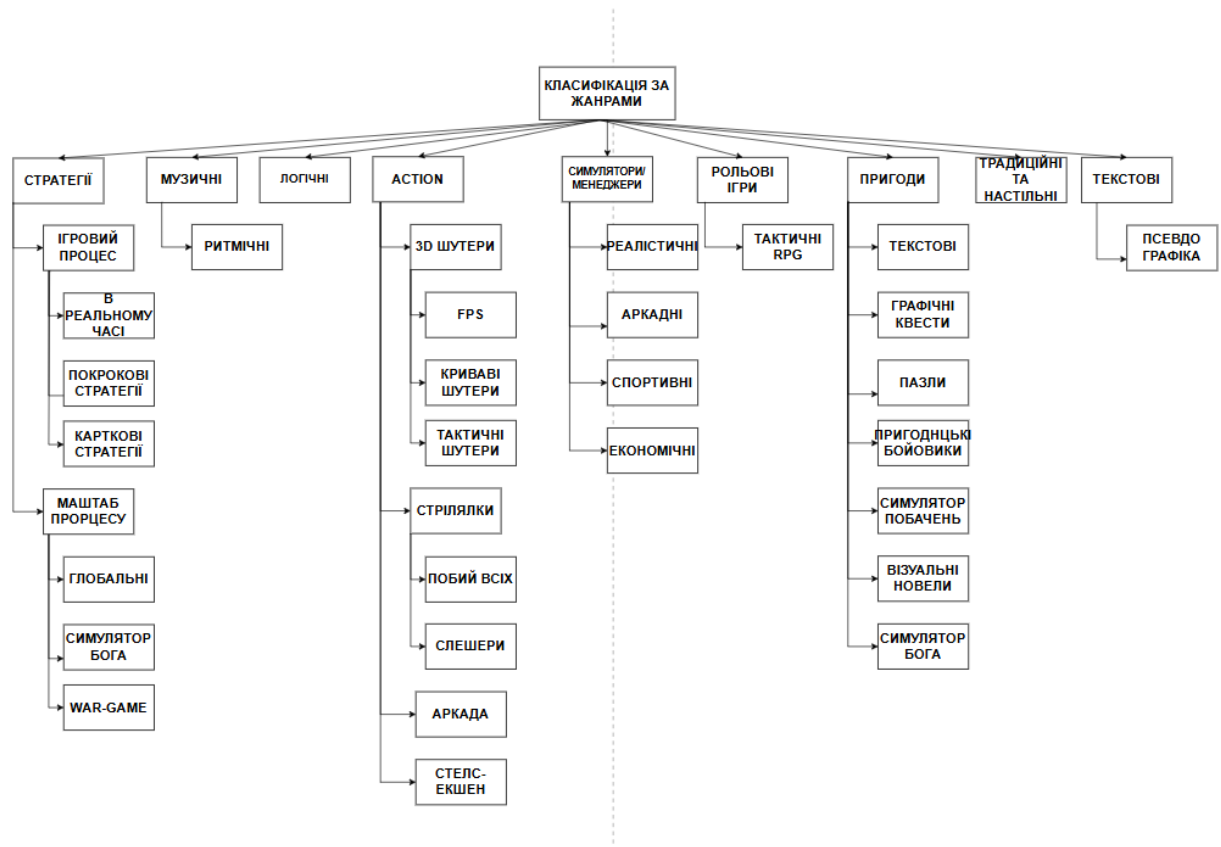


Рисунок 1.2 – Класифікація ігор [8]-[9]

Пісочниці. Термін відомий через використання в техніці, також znаний як відкритий режим. Гра концентрується не на лінійності сюжету, а на різноманітності виборів та дій. Характеризується відкритим світом та непослідовним ігровим процесом. Гра має не конкретні цілі, наприклад перемогти боса, а низку завдань, виконати які можна в різний спосіб. Така концепція дає більш захоплюючий досвід та заохочує гравця до експериментів. Все частіше до цього жанру поєднують ігри-симулятори, такі як The Sims, Minecraft чи Grand Theft Auto.

Стратегії в реальному часі. На відміну від покрокових стратегій, це змагання сам на сам в реальному часі. Ігри поєднують у собі керування ресурсами та картою. Часто мають вигляд зверху до низу. Одними з найпопулярніших RTS – ігор є: Warcraft, Age of Empires і Command & Conquer [8]-[9].

Шутери. В іграх даного типу для досягнення результату гравці мають знищувати ворогів різними видами зброї ближнього та дальнього типу. Це давній жанр, який поділився на два основні напрямки: шутери від першої особи(FPS) та шутери від третьої особи(TPS). Хоча наразі багато сучасних ігор мають змогу перемикатися між точкою зору першої та третьої особи. Окрім цього, є «shoot-em-ups» та «bullet-hell», сенс який заснований на рухливому геймплеї та великій кількості ворогів. Зараз все частіше говорять про шутери в контексті елементів в іграх інших жанрів. Прикладом такої гри є Grand Theft Auto 5.

Багатокористувацька онлайн бойова арена(MOBA). Цей піджанр стрімко набирає популярність. Він має багато зв'язків з іншими стилями та тісно пов'язаний з стратегіями в реальному часі. Головними елементами є управління ресурсами та картою, а також конкуренція та співпраця між гравцями. На відміну від RTS-ігор у MOBA ви керуєте лише одним персонажем, хоч зазвичай і маєте команду, що ускладнює побудову стратегії та прийняття рішень. Популярними іграми цього жанру є Dota та League of Legends, обидві займають значне місце в кіберспорті [8]-[9].

Рольові ігри. Суть цих ігор базується на одній простій умові: управляючи персонажем ви збираєте ресурси для прокачки персонажа. Рольові ігри є основою ігор. Жанр дуже розрісся та здобув велику кількість піджанрів. Тож його не можна охарактеризувати лише одною грою. Ось найпоширеніші піджанри:

- рольова гра: охоплює великий пласт від настільних ігор (Dungeons&Dragons) до відеоігор RPG (Fallout);
- ARPG: цей вид RPG має акцент на боях;
- CRPG: рольові ігри мають велику кількість діалогів, детально пропрацьований світ та сюжет, свободу у виконанні основних та побічних завдань. Яскравим прикладом може бути Fallout;

- MMORPG: цей піджанр поєднує в собі RPG та багатокористувацький онлайн формат. Представниками цього піджанру є World of Warcraft та Elder Scrolls Online;
- TRPG: тактичні рольові ігри. Схожі на стратегії, але акцентують увагу на креативних швидких рішеннях, як у XCOM;
- JRPG: японська RPG з деталізованим світом та діалогами, але лінійним сюжетом. Має лінійну прокачку персонажу, але дуже гарну графіку;
- roguelike: має суворо покрокову гру, процедурну генерацію та мозаїчну графіку. Нові приклади зосереджуються на діях та платформерах, таких як The Binding of Isaac та Spelunky [8]-[9].

Симулятори та спортивні ігри. Наразі спортивні симулятори настільки популярними, що мають інтеграції навіть зі спортивними організаціями – від гоночних трас до кортів. NBA 2K та Madden NFL – це відомі приклади детального відтворення професійного футболу та баскетболу. Симулятори також охоплюють деякі ігри з пісочниці, ігри віртуальної реальності та створення світу. Їх ключова задача створення реалістичного ігрового світу з захоплюючим сценарієм. The Sims та Minecraft – приклади симуляторів, які також можна віднести до пісочниць. Також вони передбачають імітацію фізичної поведінки та управління складними системами, такі як автомобілі, винищувачі, човни та інше. Аркада є спрощеною версією технічних симуляторів, в ній може бути реалізована альтернативна фізика. Принциповою відмінністю є наявність в симуляторах фізичної моделі [8]-[9].

Головоломки та ігри для компанії. Обидва жанри тісно перетинаються. Такі ігри засновані на певних правилах або є традиційними настільними іграми. Ігри для компанії розширюють це поняття включаючи мультиплеєрний режим. Можуть мати дуже просту реалізацію, як Minesweeper чи з повністю розробленим середовищем та гібридним ігровим процесом. Прикладом таких ігор є Tomb Raider. В ньому головоломки є частиною пригодницької історії та є ключовою ігровою механікою.

Бойовик-пригода. Один із найпопулярніших гібридних жанрів. Цей тип ігор зосереджений на сюжеті та боях, мають чітку механіку, яка впливає на сюжет. До цієї категорії можна віднести багато ігор. Користувачі розмежовують ці два жанри спираючись на збалансованість сюжету та функцій імітації бою. Проте, можна зазначити, що пригодницькі ігри мають окремий жанр. Часто вони належать до піджанру ігор «point-and-click». Піджанр оснований на розгадуванні загадок та головоломок від першої особи.

Ігри на виживання та жахи. Вживання та жахи часто перетинаються. Для ігор на виживання притаманне курування ресурсами та системи крафту. Усі механіки направлені на збереження життя персонажа. Жахи ж є ширшою категорією та включають в себе деякі ігри на виживання. Більшість з ігор, що мають зомбі, постапокаліпсис чи багато стрибків відносять до цього жанру.

Платформери. Платформерні ігри мають просте керування та вид з боку. Donkey Kong вважається першою грою в цьому жанрі [8]-[9].

Текстові пригодницькі та візуальні новели. Через нестачу ресурсів перші пригодницькі ігри були текстовими, проте пізніше це стало окремим жанром. Гравець взаємодіє з світом через команди, а інформацію отримує у вигляді тексту та малюнків. Текстові квести можуть мати графічну складову. Це характерно для пізніх ігор(Superhero league of Hoboken). Інша сторона текстових ігор це візуальні романи. Характерна демонстрацією історії, часто романтичного жанру. В деяких варіантах гравець має змогу впливати на хід історії.

Також, ігри можна класифікувати за:

- ігровими платформами;
- кількістю гравців(одно- та багатокористувацькі);
- умовами розповсюдження(безкоштовні та платні);
- за способом розповсюдження.

З розвитком індустрії зростає кількість нових стилів, старі стилі зливаються між собою в пошуках нових рішень. Це створює перешкоду для класифікації ігор, проте дає різноманітність в ігровому процесі [10].

1.4 Аналоги комп'ютерних ігор з генерацією мапи

Наразі процедурна генерація є широко використовуваною у ігровому світі. Вона дає змогу створювати величезні світи з мінімальними ресурсними витратами. Такий алгоритм надає грі змогу надовго залишатися актуальною для користувачів. Світ створений за цим методом буде новим та непередбачуваним не зважаючи на те скільки ви граєте п'ятдесят чи п'ятсот годин.

Процедурну генерацію можна використовувати з різним підходом від процедурного створення підземель (Enter the Gungeon) і до процедурного створення ворогів та сюжету (Wilderness, Middle-Earth: Shadow of War) [11].

1.4.1 Spelunky

Це відеогра створена Дерекком Ю. та випущена у вигляді безкоштовної програми з відкритим кодом. Головним героєм є спелеолог, граючи за якого гравець повинен досліджувати печери, борючись з ворогами та збираючи скарби. В даній грі процедурна генерація використовується під час генерації печер, випадковість є ключовим елементом у створенні структури рівнів.

У цій грі генерація починається з сітки кімнат 4 на 4. Кімнати в свою чергу мають чітко побудовану структуру та шаблони. Всередині кімнати є статичні блоки та додаткові які генеруються варіативно. Така структура дає ілюзію змінної карти, яка залишається керовано обмеженою та контрольованою [12].

1.4.2 Deep Rock Galactic

Гра розроблена данською компанією Ghost Ship Games. Гра є Кооперативним шутером від першої особи. Гра переносить нас на планету Хоксес 4, яка багата на мінерали. Гравець грає за шахтаря, якого відправили у печери планети для добування ресурсів. Планета має ворожу природу, тож в процесі гравець має боротися з інопланетянами [13].

Гра цікава тим, що за допомогою процедурної генерації створює повноцінну 3D систему шахт. Також шахти мають 10 різних типів біому від заморожених і до радіоактивних. Генерація є послідовною по одній кімнаті за раз. Загальна система печер є результатом унікальної генерації, проте печерні кімнати створені вручну. Проте більшість із цих шаблонів також мають кілька внутрішніх «рандомайзерів», що унеможлиблює повтор шахти. Тип місій також впливає на генерацію карти, кімнати можуть деформуватися, перекриватися або віддзеркалюватися. Генерація починається із створення унікального seed, який впливає на всю подальшу генерацію. Печера генерується у сфері. Спочатку створюються зовнішні та внутрішні межі печери, це шар на який генератор наносить шум. Після додаються зони в яких випадковим чином додаються кам'яні утворення. На останньому етапі додаються області потенційно призначені для виходу з кімнати, вони являють собою утрамбований ґрунт. Після створення загального скелету він наповнюється шумом та «сміттям». Шум являє собою рельєф карти, а сміття є флорою та фауною. Ці два параметри є змінними і залежать від обраного типу біому [13].

1.4.3 Enter the Gungeon

Відеогра розроблена компанією Dodge Roll у жанрі «bullet hell roguelike». Гра розповідає про кількох персонажів, які спускаються на дно Підводного світу шукаючи зброю, яка може «вбити» минуле. Проходячи кімнати вони борються з ворогами у вигляді куль за допомогою різної зброї.

У грі присутня генерація карт. На перший погляд її механіка є досить простою – об'єднання випадкових кімнат зі з'єднаннями між собою. Проте розглянувши це детальніше можна побачити, що боси та вороги завжди розміщуються на оптимальній відстані, кімнати відпочинку та магазини теж доречно чергуються. Не можна не помітити сплановане розміщення скринь зі скарбами, які знаходяться у кінці циклів з односторонніми коридорами чи заховані глибоко в середині рівнів. Секретом цього є заздалегідь створені макети [14] - [15].

Такі схеми називаються потоками (рис. 1.3). Для кожного рівня гра має декілька таких потоків (щонайбільше 8).

Кожен такий набір розроблений навколо якоїсь особливості. Це може бути петля або багатостороннє розгалуження чи шлях який обов'язково проходить через магазин, тощо.

Карти мають графову структуру, а кожна кімната має свої метадані. До загальної структури вводяться додаткові вузли. Так звана ін'єкція має данні що визначають який предмет, куди і як треба вставити, ймовірність появи чи її умови. Наприклад, секретна кімната не має умов появи. Шанс на її появу становить 90%, а з'являється вона зазвичай у глухих кутах, але має 1/5 шансу на приєднання до будь-якої кімнати. В кінці для кожної кімнати генеруються вороги та наповнення [14]-[15].

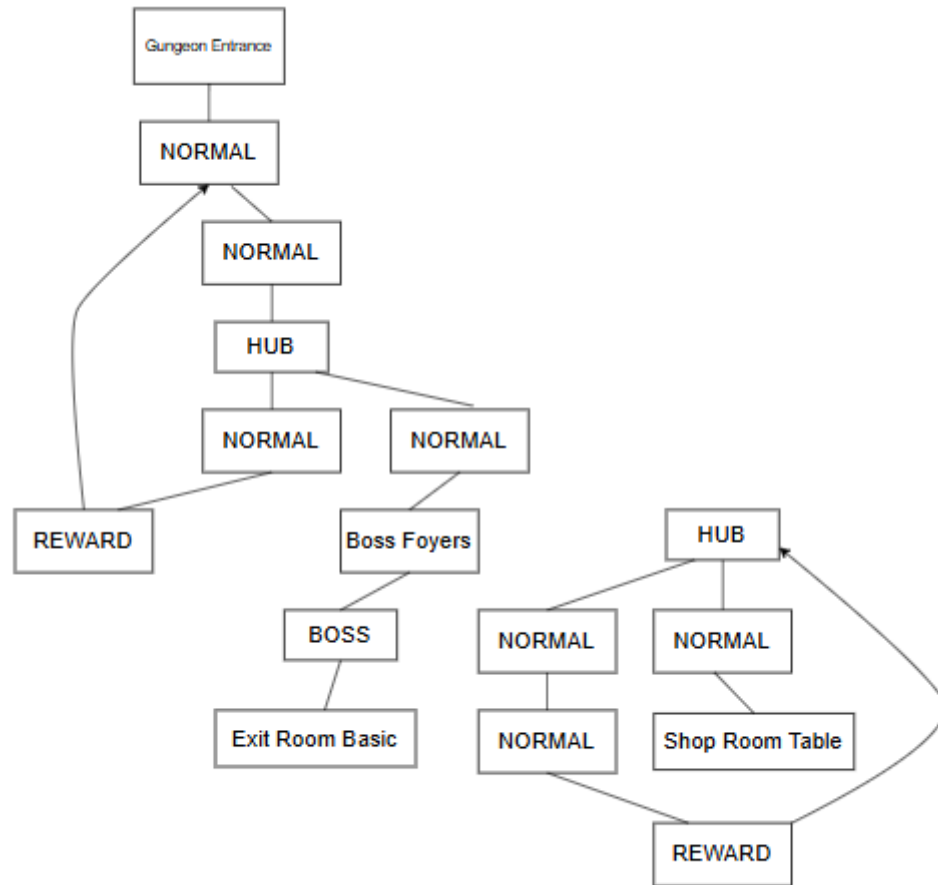


Рисунок 1.3 – Схема рівня [15]

1.4.4 Invisible, Inc.

Це покрокова тактична гра, в якій гравець є частиною шпигунського агентства, яке зазнало нападу. Гру випустила у 2015 році компанія Klei Entertainment. Для проходження гри гравець має виконувати місії контролюючи агентів. Під час проходження місій йому потрібно знищувати ворогів, проникати у кімнати, зламувати сейфи та виконувати приховані завдання. Завдяки генерації не тільки рівнів, а й завдань, гра забезпечує неповторний досвід при кожному проходженні. Через велику кількість генерованих даних під час проходження можуть з'являтися збої, як от вороги згенеровані в некоректному місці та блокування прогресу гри чи таємних приміщень через

це. Під час генерації використовуються шаблони кімнат, кімнати мають свої пріоритети та поєднуються на основі сітки та враховуючи доступність. В кінці генеруються вороги. [16]

Отже, процедурна генерація є гарним засобом для створення різноманіття у грі. Вона часто використовується маленькими компаніями для збереження ресурсів. У кожній з ігор ми можемо побачити комбінацію процедурної генерації з іншими методами такими як випадкова генерація об'єктів чи неігрових персонажів.

У таблиці 1.1 порівнян основні характеристики оглянутих ігор.

Таблиця 1.1 – Порівняння оглянутих ігор

Характеристика	Spelunky	Deep Rock Galactic	Enter the Gungeon	Invisible, Inc.
Тип генерації	Кімнати та вороги	Повноцінна 3D мапа	Мапа рівня, вороги	Локації, цілі, перешкоди та складність місії
Алгоритм	Власний алгоритм(поєднання шаблонної генерації з сіткою)	Wave функція, графова топологія, seed-based генерація	Графова генерація	Модифікований алгоритм розміщення кімнат на сітці
Рівень варіаційності	Високий	Висока	Середня	Високий
Глибина інтерактивності	Середній(бомби)	Висока(копання)	Середня	Високий(стелс)
Візуальний стиль	2D	Низькополігональний 3D	2D піксель-арт	2D-ізометрія
Платформи	Windows, Nintendo Switch, PS 4, PS 3, ChromeOS, Xbox	Windows, PS 4, , PS 3, Xbox	Windows, Linux, PS 4, iOS, Nintendo Switch, Android, macOS, Xbox, Google Stadia	Windows, OS X, Linux, PS 4, iOS, Nintendo Switch

1.5 Постановка завдань дипломної роботи

Розглянуті ігри демонструють, що процедурна генерація є популярною технологією, яка здатна значно зекономити ресурси проєкту. Вона використовується у різних жанрах і має широкий спектр використання від генерації мап та рівнів до генерації сюжетів та персонажів. Комбінація процедурної генерації з іншими методами дає більш контрольовані результати та значно зменшує вірогідність багів. Популярним підходом є генерація за допомогою кімнат, створених вручну. В таких рішеннях процедурна генерація відповідає за створення загального скелету карти та зв'язків між кімнатами.

Під час аналізу було детально вивчено процес створення ігор. Хоч він і складається із загальних 7 етапів розробки, всередині ж в об'єднанні великої кількості елементів таких як: геймдизайн, графіка, звукове оформлення, програмування, дизайн рівнів та інші аспекти. Ігрові рушії відіграють важливу роль у цьому процесі спрощуючи та пришвидшуючи створення 2 та 3D проєктів. Отже отримані результати дозволяють сформулювати чітке уявлення про обрану область. Це доводить доцільність реалізації гри з процедурною генерацією, як інноваційного й актуального проєкту в ніші інді-ігор. Також було сформовано загальні вимоги до гри з процедурною генерацією.

Мета роботи – програмна реалізація алгоритму побудови лабіринту для гри з механікою динамічної генерації мапи.

Завдання роботи:

- дослідити існуючі алгоритми генерації ігрової мапи та обґрунтувати вибір одного з них для реалізації ігрового застосунку з процедурно згенерованим середовищем;
- виконати проєктування та конструювання програмного забезпечення;
- розробити інструкцію з використання програми.

2 ДОСЛІДЖЕННЯ АЛГОРИТМІВ ГЕНЕРАЦІЇ ІГРОВОЇ МАПИ

2.1 Огляд та класифікація лабіринтів

Лабіринт – будь-яка структура, яка складається з мережі шляхів, проходів та тупиків. На початку – споруда чи відкритий простір з безліччю залів, кімнат коридорів та добудов, які склалися у заплутані маршрути аби збити з тями та не дати вийти недосвідченим людям [17].

Лабіринти класифікують за сімома параметрами:

- мірність;
- гіперрозмірність;
- топологія;
- мозаїка;
- маршрутизація;
- текстура;
- фокус.

Клас мірність визначає кількість вимірів у просторі в який існує лабіринт та поділяється на [18]:

- 2D: більшість лабіринтів є двовимірними. Такий лабіринт можна зобразити на аркуші паперу не перекриваючи інші проходи;
- 3D: лабіринт, який складається з кількох рівнів та може мати проходи вгору та вниз. Часто зображають масивом двовимірних рівнів;
- багатовимірні: такі лабіринти мають від чотирьох вимірів. Іноді зображуються як 3D-лабіринт з спеціальними «порталами» для переміщення між вимірами;
- переплетені: двовимірний або 2.5-вимірний лабіринт проходи якого можуть перекриватися між собою.

Клас гіперрозмірність описує вимірність об'єкта, який рухається лабіринтом:

- негіперлабіринт: майже всі лабіринти. Об'єктом є невелика куля або ви самі. Об'єкт переміщується з точки в точку, а його шлях формує лінію. Кількість варіантів у кожній точці є скінченним та може бути обрахована [18];

- гіперлабіринт: Це тривимірний або вищевимірний лабіринт. Головна відмінність цього лабіринту в об'єкті. Він являється не просто точкою, а лінією. Шлях позаду такого об'єкта утворює поверхню. В такому лабіринті треба працювати одразу з усією лінією для запобігання перетворення її у точку, яку можна розглядати як негіперлабіринт;

- гіпергіперлабіринт: друга назва гіперлабіринт другого порядку. Такий лабіринт існує в чотиривимірному та вищевимірних просторі. Об'єктом такого лабіринту є площа, а позаду неї утворюється тверде тіло.

Клас топологія описує просторову геометрію в якій існує лабіринт. Вони поділяються на:

- звичайний: це звичайний лабіринт, який існує в евклідовому просторі;

- планарний: будь-який лабіринт з аномальною топологією. Такі лабіринти мають незвичні з'єднання країв. Наприклад, лабіринт на поверхні куба [18].

Мозаїка відповідає за певні клітини, які утворюють лабіринт:

- ортогональна: лабіринти з прямокутною сіткою. Всі перетини відбуваються під прямим кутом;

- дельта: лабіринт являє собою трикутні кімнати, що мають до трьох проходів;

- сігма: лабіринт являє собою переплетені шестикутники, що мають до шести проходів;

- тета: лабіринт складається з концентрованих кіл. Один кінець шляху знаходиться на зовні, а інший в центрі;

- епсілон: лабіринт складається з восьмикутників та квадратів.

- зета: має сітку, як в ортогонального лабіринту, однак проходи можуть бути не лише під кутом 90, а й 45 градусів;
- омега: має відношення до лабіринтів з послідовною неортогональною теселяцією. Наприклад, дельта та сігма підпадають під цю класифікацію;
- фрактальний: лабіринт, що має фрактальну структуру. Він складається з менших повторюваних лабіринтів;
- тріщини: лабіринт є аморфним. Його стіни розташовані під випадковим кутом [18].

Маршрутизація. Клас описує типи проходів у геометрії:

- ідеальний: лабіринт, який немає петель та недоступних ділянок. Лабіринт має одне рішення та є однозв'язним;
- коса: лабіринт, який не має глухих кутів. Його проходи зливаються та змушують блукати колами;
- унікальний: лабіринт, який не має перехресть. Має єдиний звивистий шлях. Є дуже простим;
- розріджений: такий лабіринт має недоступні частини. Його проходи можуть формуватися не через кожну клітинку;
- часткове плетіння: лабіринт з петлями та глухими кутами [18].

Текстури є загальними характеристиками, які описують стиль пасажів у трасах геометрії. Вони можуть бути упередженими, тобто такий лабіринт має великі прямі ділянки які йдуть в одному напрямку. Симетричні лабіринти мають симетричні проходи. Фактор бігу характеризує наскільки довгий шлях без поворотів ми маємо. Чим більше цей показник тим довгими є проходи.

Клас фокус ділиться на два типи: додавання стін та створення проходів. Вони являють собою різницю під час генерації та не впливають на візуальну різницю. У першому варіанті створення починається з порожнього поля, яке заповнюється стінами. У другому – навпаки поле є суцільним блоком з якого екструдуються стіни [18].

2.2 Дослідження існуючих методів генерації лабіринтів

У сучасному світі лабіринти з простих головоломок трансформувалися у дещо більше та стали частиною багатьох галузей таких як: архітектура, робототехніка, ігри чи текстиль. Такий попит спричинив розвиток методів створення лабіринтів. Наразі існує понад 20 методів генерації лабіринтів. Багато з них мають модифіковані версії. Також для отримання кращих результатів методи почали поєднувати створюючи гібриди. Таке різноманіття може викликати труднощі під час підбору алгоритму для конкретної задачі.

Під час підбору алгоритму ми маємо зважати на багато факторів таких як : тип бажаного лабіринту, складність реалізації, продуктивність, випадковість, складність лабіринту та його візуальна складова. Розглядаючи ігри ми маємо додаткові параметри у вигляді генерації на основі сітки чи генерації у 3D просторі [19].

Ідеальним лабіринт вважають якщо кожна пара кімнат формує лише один шлях. Ідеальним алгоритмом же вважають той, що генерує всі можливі варіанти з рівномірною ймовірністю. Такі алгоритми є неупередженими.

Неупереджені лабіринти часто будують схожі лабіринти, які не надто підходять для ігор.

Для порівняння було обрано 5 основних методів, які найкраще підходять для використання у контексті відеоігор:

- Recursive Backtracker;
- Kruskal`s Algorithm;
- Aldous-Broder Algorithm;
- Hunt-and-kill Algorithm;
- Binary Tree Algorithm.

Recursive Backtracker. Інша назва модифікований пошук у глибину. Сенс алгоритму доволі простий та ефективний. Етапи роботи алгоритму виглядають наступним чином (рис. 2.1).

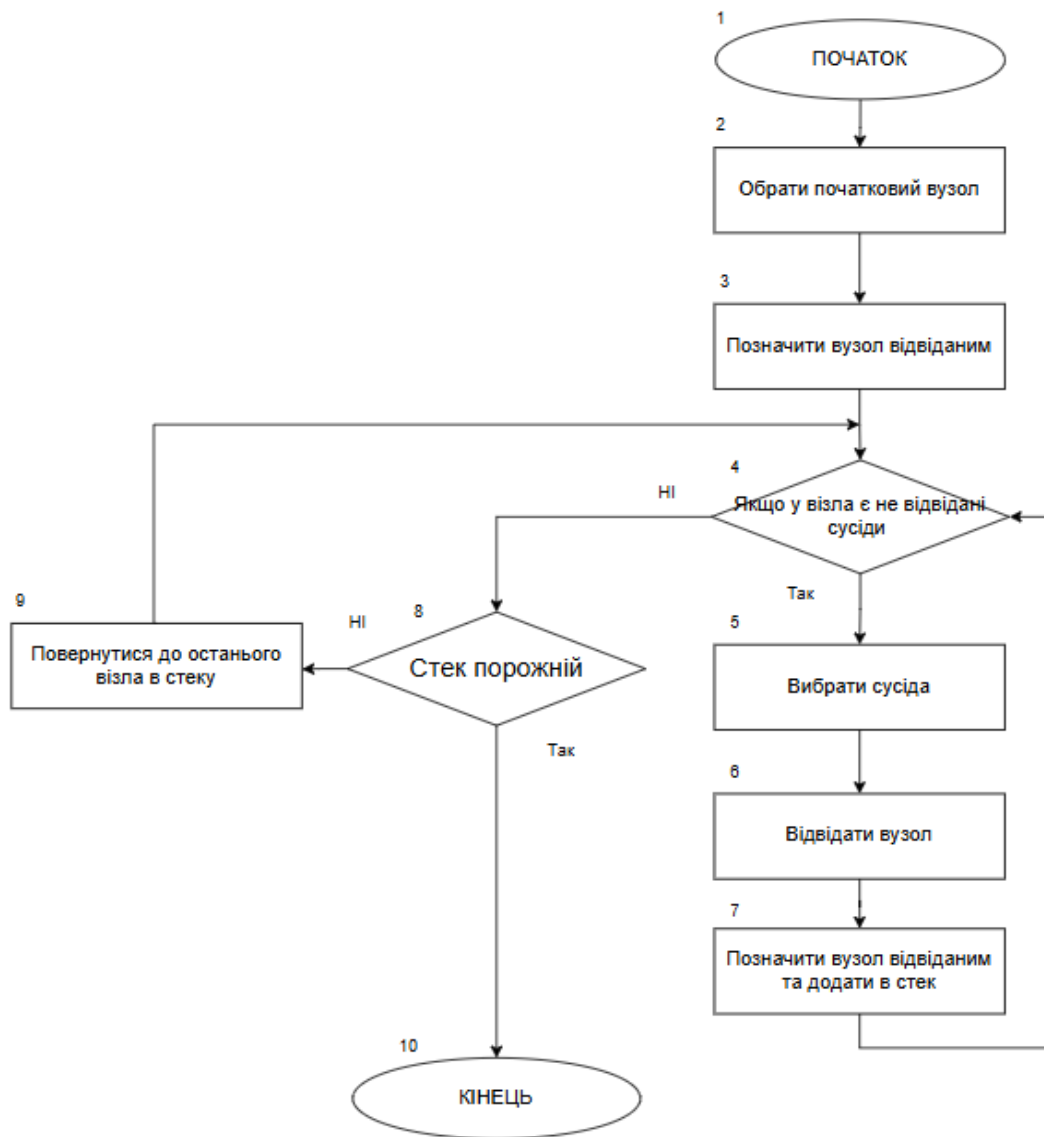


Рисунок 2.1 – Схема алгоритму Recursive Backtracker

Блок 1. Початок: ініціалізація алгоритму генерації лабіринту.

Блок 2. Обрати початковий вузол: випадково або заздалегідь визначається стартова клітинка, яка стане першою у лабіринті.

Блок 3. Позначити вузол відвіданим: поточна клітинка вважається частиною лабіринту. Вона додається до списку відвіданих.

Блок 4. Перевірити наявність невідвіданих сусідів: для поточної клітинки перевіряються всі можливі сусіди (вгору, вниз, ліво, право), що ще не були включені до лабіринту.

Блок 5. Якщо є невідвідані сусіди — вибрати одного: з доступних напрямків випадковим чином обирається один напрямок.

Блок 6. Відвідати сусідній вузол: ламається стіна між поточним вузлом і обраним сусідом.

Блок 7. Позначити новий вузол відвіданим і додати до стеку: обраний вузол додається до стеку, і алгоритм повторюється з нього.

Блок 8. Якщо немає невідвіданих сусідів — перевірити стек: якщо у вузла більше немає напрямків для руху, відбувається повернення до останнього вузла у стеку.

Блок 9. Повернення назад: поступово повертаємося назад по шляху до тих пір, поки не знайдеться новий вузол з доступними сусідами.

Блок 10. Кінець: алгоритм завершується, коли стек порожній, і всі клітинки, які могли бути відвідані, вже опрацьовані [19]–[21].

Цей алгоритм не генерує рівномірно охоплююче дерево. Проте він є простим та швидким, а лабіринти які він генерує є доволі «адекватними».

Часова складність алгоритму складає $O(n)$.

Потребує пам'яті для стеку рекурсії.

Серед переваг проста та легка реалізація. Серед недоліків створення лабіринтів з довгими коридорами [19]–[21].

Kruskal's Algorithm. Мінімальний алгоритм охоплюючого древа, який створює ідеальний лабіринт(рис. 2.2).

Етапи роботи алгоритму виглядають наступним чином.

Блок 1. Початок: ініціалізація алгоритму.

Блок 2. Створити список стін: Кожна клітинка розглядаються як окрема множина. Стіни додаються у загальний список.

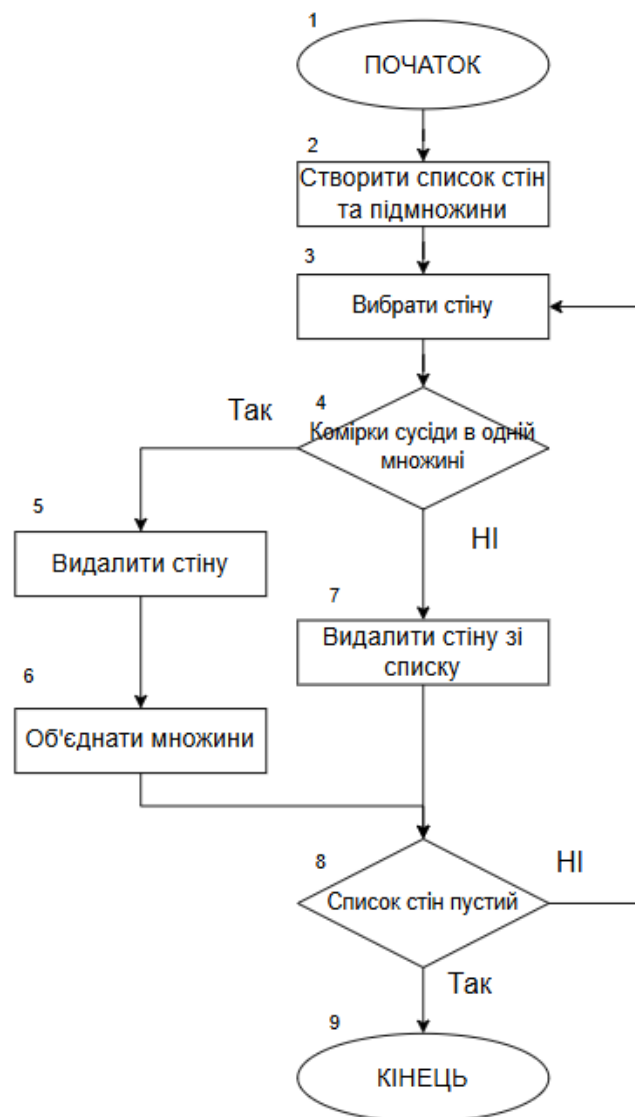


Рисунок 2.2 – Схема алгоритму Kruskal's Algorithm

Блок 3. Випадково вибрати стіну зі списку: Випадково обирається одна з доступних стін.

Блок 4. Перевірити множини клітин по обидва боки стіни: якщо обидві клітинки належать до різних підмножин – продовжуємо.

Блок 5. Видалити стіну з мапи: Візуально видаляється бар'єр між клітинками – утворюється прохід.

Блок 6. Об'єднати множини клітинок: Об'єднати множини двох клітинок у спільну, оскільки тепер між ними є прохід.

Блок 7: Клітинки в одній множині: Якщо обидві клітинки вже в одній множині – це утворить цикл. Тому стіну не використовуємо, але прибираємо зі списку.

Блок 8. Чи порожній список: якщо в списку більше немає елементів – алгоритм завершено.

Блок 9. Кінець: побудова лабіринту завершена – всі клітини пов’язані.

Часова складність алгоритму складає $O(E \log E)$.

Пам’ять задіяна для збереження наборів.

Перевагами алгоритму є рівномірне розділення шляхів лабіринту. Недоліком є великі обчислювальні витрати для великих лабіринтів.

Aldous-Broder Algorithm. Це алгоритм генерації рівномірної основи для неорієнтованого графа. Теоретично алгоритм, який генерує рівномірне дерево, – це єдиний алгоритм, яким може побудувати кожен можливий лабіринт. Така властивість робить алгоритм значно неефективнішим[19]–[21]. Етапи роботи алгоритму виглядають наступним чином (рис. 2.3).

Блок 1. Початок: ініціалізація алгоритму.

Блок 2. Вибрати клітинку: випадковим чином вибирається початкова клітинка лабіринту.

Блок 3. Клітинка відвідана: перевіряється, чи була поточна клітинка вже відвідана. Якщо так – вибирається нова, якщо ні алгоритм іде далі.

Блок 4. Позначити клітинку як відвідану: поточна клітинка помічається як відвідана.

Блок 5. З’єднати з попередньою клітинкою: якщо клітинка нова, вона з’єднується ребром з попередньою, утворюючи частину дерева.

Блок 6. Усі клітинки відвідані: перевіряється, чи були відвідані всі клітинки. Якщо так алгоритм завершується, якщо ні повертається до вибору нової клітинки.

Блок 7. Генерація лабіринту завершена.

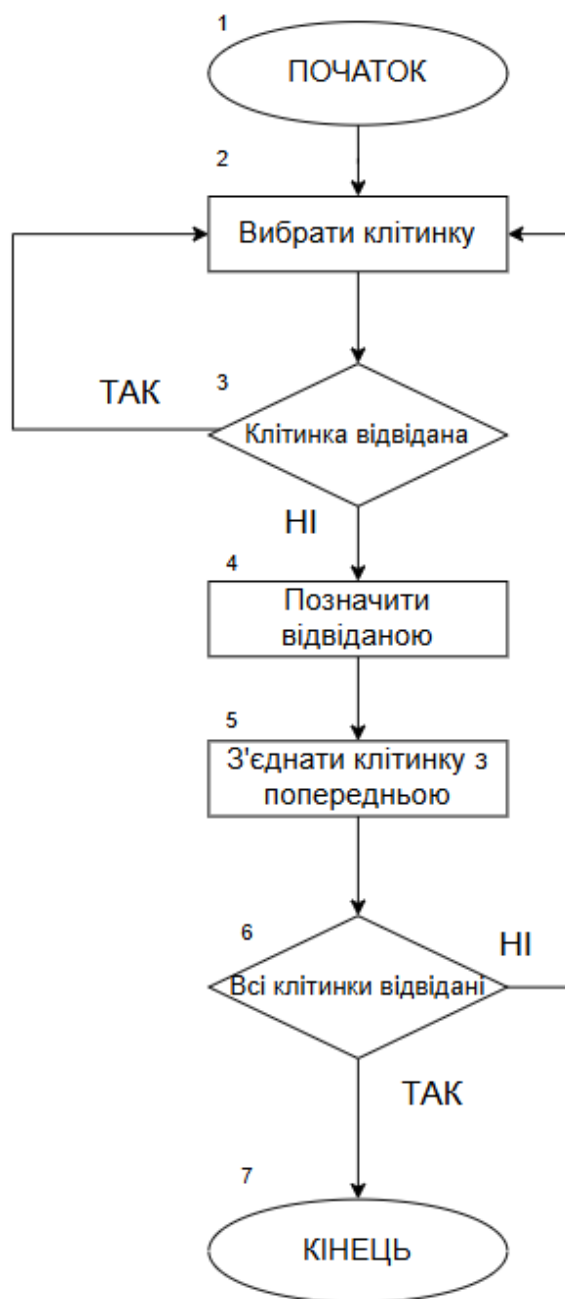


Рисунок 2.3 – Схема алгоритму Aldous-Broder Algorithm

Його неефективність полягає в тому, що алгоритм може дуже довго шукати останні невідвідані клітинки. У найгірших випадках – взагалі не знайти. Проте створення рівномірного основного дерева є його унікальною властивістю. Окрім нього такою властивістю володіє лише один алгоритм [19]–[21].

Часова складність алгоритму складає $O(n^2)$.

Зберігання станів лабіринту потребує додаткової пам'яті.

Плюсом є простота та безупинність процесу. Варіативність форм лабіринту теж є доволі великою. З мінусів в деяких випадках алгоритм може бути неефективним та вимагати великої кількості ресурсів.

Hunt-and-kill Algorithm. Цей алгоритм схожий з Recursive Backtracker, але на відміну від нього виконує ітеративний пошук. Етапи роботи алгоритму виглядають наступним чином (рис. 2.4).

Блок 1. Початок: ініціалізація алгоритму.

Блок 2. Вибрати клітину: випадковим чином вибирається клітинка. Починається режим «вбивства».

Блок 3. Відмітити клітинку: помітити обрану клітинку як відвідану.

Блок 4. Невідвідані сусіди: якщо поточна клітинка має сусідів, які ще не були відвідані продовжується режим «вбивства», якщо ні – режим «полювання».

Блок 5. Вибрати сусіда: випадковим чином вибрати одного з невідвіданих сусідів.

Блок 6. Видалити стіну: видалити стіну між поточною клітиною та вибраним сусідом. Перейти до нового вузла.

Блок 7. Пошук клітини з невідвіданим сусідом: запускається режим «полювання» - перегляд усіх клітинок (зліва направо), поки не буде знайдено таку, що вже була відвідана та має хоч одного невідвіданого сусіда.

Блок 8. Клітинка знайдена: якщо клітинка знайдена то перейти до неї і продовжити режим «вбивства». Якщо ні – алгоритм завершений.

Блок 9. Кінець: усі клітини оброблено, лабіринт згенеровано.

Часова складність алгоритму складає $O(n)$. Пам'ять задіяна для зберігання поточних координат та стану [19]–[21]. Його лабіринт може мати відкритий простір. Також має збалансовану випадковість, проте це може призводити до менш передбачуваних лабіринтів.

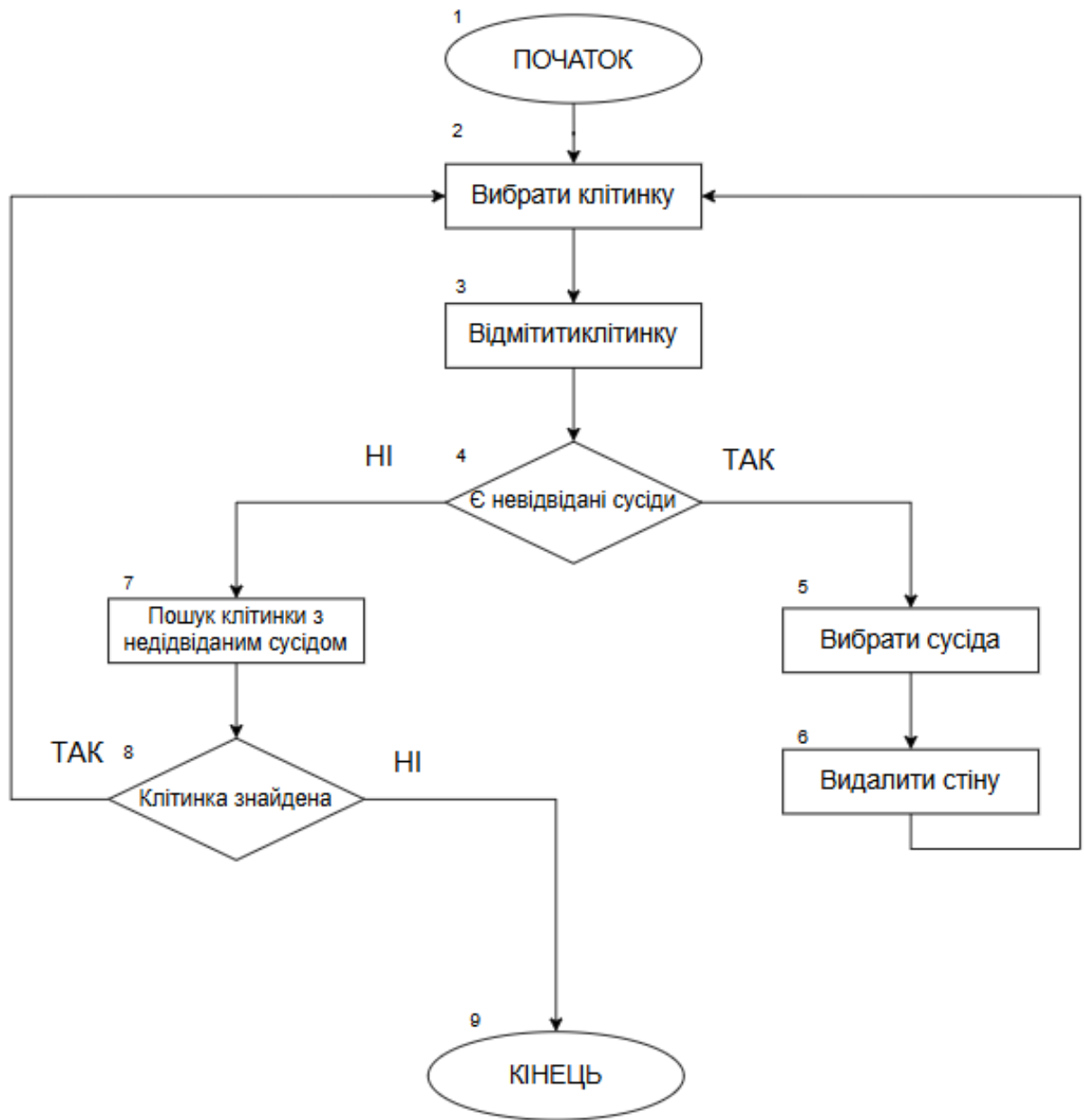


Рисунок 2.4 – Схема алгоритму Hunt-and-kill Algorithm

Binary Tree Algorithm. Є одним із найпростіших алгоритмів. Етапи роботи алгоритму виглядають наступним чином (рис. 2.5).

Блок 1. Початок: ініціалізація алгоритму.

Блок 2. Вибрати клітинку: перехід до чергової клітинки. Алгоритм проходить кожну клітинку сітки послідовно в одному з наборів діагоналей(північ/захід, північ /схід, південь/захід чи південь/схід).

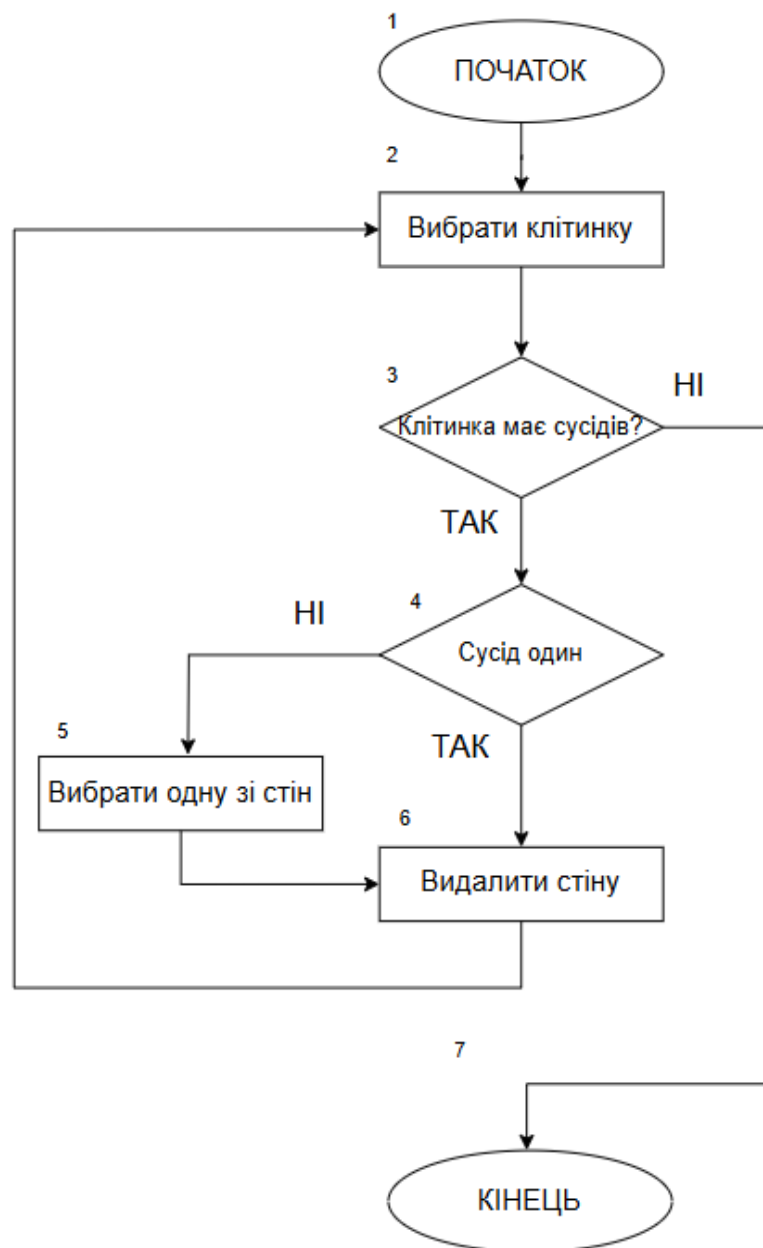


Рисунок 2.5 – Схема алгоритму Binary Tree Algorithm

Блок 3. Визначити доступні напрямки: перевірити, чи можна видалити стіну в одному з двох дозволених напрямків.

Блок 4. Випадково вибрати напрямок: випадковим чином обрати один із доступних напрямків.

Блок 5. Видалити стіну: видалити стіну між поточною клітинкою та сусідом у вибраному напрямку.

Блок 6. Клітинки пройдено: якщо залишилися ще не пройдено клітинки то повернутися до пункту 2, якщо пройдено – алгоритм завершено.

Блок 7. Кінець: алгоритм завершено.

Доволі проста та швидка, але його лабіринти мають сильний діагональний ухил [19]–[21].

Часова складність алгоритму складає $O(n)$. Пам'ять задіяна для зберігання поточних координат та стану. Максимально простий та легкий в реалізації, при цьому швидкий та ефективний. Проте шлях є зміщеним та має лише один напрямок, що спричиняє обмеження в варіативності лабіринту [19]–[21].

Результати порівняння наведено в таблиці 2.1.

Таблиця 2.1 – Порівняння алгоритмів

Характеристика	Kruskal's	Aldous-Broder	Hunt-and-kill	Binary Tree	Recursive Backtracker
Часова складність	$O(E \log E)$	$O(n^2)$	$O(n)$	$O(n)$	$O(n)$
Тип лабіринту	Ідеальний	Ідеальний	Ідеальний	Неідеальний	Ідеальний
Тип алгоритму	Мінімальне остовне дерево	Стохастичний	Гібридний пошук	Жадібний пошук	Пошук глибини
Випадковість	Висока	Дуже висока	Середня	Низька	Висока
Складність реалізації	Висока	Низька	Низька	Дуже низька	Середня
Шлях лабіринту	Розгалужений	Рівномірний	Звивистий	Прямолінійний, прогнозований	Звивистий
Придатний для 3D	Так	З ускладненнями	Так	Так	Так
Створює довгі коридори	Помірно	Ні	Помірно	Ні	Так

З наведеного можна зробити висновок що найкращим варіантом буде зворотна рекурсія. Вона створюватиме цікаві для гравця та не шаблонні лабіринти. При цьому не потребує надмірного використання даних та доволі швидка в роботі. Серед мінусів можна виділити створення довгих коридорів, що може погіршити ігровий досвід. Також непоганим варіантом може бути алгоритм полювання та вбивства. Він має дещо нижчу варіативність генерації, проте не має довгих коридорів та трохи простіший у реалізації. Однією з його особливостей є створення відкритого простору, що може бути недоречним у деяких іграх з лабіринтами.

2.3 Висновки за розділом 2

Поняття лабіринту та методів його генерації є ключовими елементами у побудові ігор пригодницького та логічного жанрів. Основну увагу було приділено класифікації лабіринтів, їх характеристикам, властивостям та топології. Також було проаналізовано п'ять алгоритмів генерації. Проаналізовано їх складність та ефективність.

Для порівняння було вибрано п'ять методів, які найчастіше використовуються під час створення ігор: Recursive Backtracker, Kruskal's Algorithm, Aldous-Broder Algorithm, Hunt-and-kill Algorithm та Binary Tree Algorithm. Усі ці алгоритми мають свої переваги, недоліки та особливості.

Найкращими алгоритмами для поставленої задачі стали Recursive Backtracker та Hunt-and-kill. Вони забезпечують високу варіативність та ефективність у роботі. Придатні як для 2D так і для 3D ігор та мають високу продуктивність.

Отже, отримані результати дозволили детально дослідити тему та отримати ширші знання у темі лабіринтів. Це дає змогу краще розібратися у побудові проєкт та обґрунтувати вибір алгоритму генерації мапи для гри та його реалізації.

3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз вимог до системи

Було проаналізовано технічне завдання (Додаток А) та виділено основні вимоги до програми.

Розроблена ігрова програма в жанрі головоломок повинна мати зручний інтерфейс, бути оптимізованою для ПК, а також забезпечити інтуїтивно зрозумілий геймплей.

Програма повинна реалізувати наступні функції:

- переміщення персонажа у 3D просторі;
- генерація лабіринту;
- реалізація механіки виграшу;
- розміщення об'єктів у лабіринті;
- налаштування гри.

3.2 Розробка архітектури системи

Для реалізації проєкту буде використовуватися Singleton шаблон. Особливістю цього шаблону є те, що клас має лише один екземпляр протягом усього виконання програми та забезпечує глобальний доступ до нього.

Основна ідея Singleton полягає в тому, що замість створення багатьох об'єктів одного класу чи передавати їх іншим об'єктам, створюється лише один екземпляр класу, який доступний всюди через статичне поле. Таким чином ніхто окрім самого класу не може замінити кешований екземпляр.

Цей шаблон надає можливість простої реалізації глобального доступу до об'єктів, зручне збереження станів між сценами та централізоване управління даними.

Серед основних мінусів шаблону можна виділити приховані залежності, які можуть ускладнити тестування та масштабування проєкту.

Також Singleton потребує багатопотоковості аби запобігти створення декількох екземплярів різними потоками.

У Unity Singleton широко використовується при створенні об'єктів управління (Manager), які повинні існувати лише в одному екземплярі. Цей шаблон зазвичай реалізується на основі MonoBehaviour. У методі Awake() відбувається перевірка на існування екземпляра об'єкта. Якщо екземпляр існує дублікат знищується, якщо ні – він призначається та зберігається за допомогою DontDestroyOnLoad() [22].

Також для зручної підтримки розширення та підтримки буде використано модульний підхід. Він дозволить розділити всі ключові елементи гри на окремі скрипти.

Всі візуальні елементи сцени будуть реалізовані за допомогою prefab-об'єктів. Prefab – це шаблон ігрового об'єкта, який містить Game Object з усіма його додатковими налаштуваннями. Такий об'єкт можна багаторазово використовувати на сцені або створювати динамічно у коді [23].

3.3 Порівняння ігрових рушіїв

Ігровий рушій – це основна програмна частина гри, яка складається з програмних модулів та інструментів для створення й запуску інтерактивних графічних застосунків із обробкою в реальному часі [24].

Для порівняння було взято найпопулярніші ігрові рушії для створення відеоігор для персональних комп'ютерів: Unity, Unreal Engine та Gotot.

3.3.1 Unity

Unity – це кросплатформний ігровий рушій створений компанією Unity Technologies. Цей рушій є одним із найпопулярніших та добре підходить новачкам. Він підтримує більшість цільових платформ та підтримує 2D та 3D

розробку, а також пропонує широкий спектр інструментів для різних жанрів та платформ, що робить його універсальним. Його можливості дозволяють створювати додатки, які працюватимуть на понад 25 платформах. Unity має велику спільноту та гарно прописану документацію, а її сховище активів є найбільшим серед рушіїв. Також Unity має безкоштовну персональну версію, що є чудовим рішенням для створення невеликих інді-ігор.

Для розробки гри на рушії Unity можна використовувати C# або JavaScript. Редактор має простий інтерфейс з підтримкою Drag & Drop. Він складається з кількох робочих вікон, які легко налаштовуються, що дозволяє налагоджувати гру безпосередньо в середовищі. Також у сучасних версіях доступне програмування за допомогою Visual Scripting, що дозволяє створювати ігрову логіку без програмування [24]–[26].

У Unity є підтримка системи Level of Detail – це система, яка автоматично замінює складні моделі на простіші, якщо вони знаходяться далеко від гравця. Та Occlusion Culling, яка виключає з рендеру об'єкти поза полем зору камери. Це значно зменшує навантаження на CPU та покращує продуктивність.

3.3.2 Unreal Engine

Unreal Engine – рушій від Epic Games, який є головним конкурентом Unity. Він став результатом комерціалізації ігрового рушія, який використовувався всередині компанії.

Вважається, що Unreal має найкращі графічні можливості та дозволяє створювати фотореалістичні ігри. Він широко використовується для створення високоякісних 3D-ігор, особливо в індустрії AAA-ігор [26]–[27].

Для розробки ігор використовується C++, але він також поставляється з візуальним інтерфейсом сценаріїв під назвою Blueprint – це інтуїтивно зрозуміла система створення логіки 3D-середовищ. Користувач може

створювати об'єкти певних типів та розміщувати їх на сцені. Вся логіка може бути розроблена без будь-якого програмного коду, що спрощує процес розробки для користувачів, які не мають глибоких знань у програмуванні.

Останні версії рушія включають передові технології в галузі тривимірної графіки, серед яких: системи глобального освітлення Unreal Lightmass, засоби постобробки зображення, реалістична фізика, анімація персонажів та інструменти для створення великих ландшафтів та багато іншого. Рушій є безкоштовним, але вимагає сплати 5% від загальних продажів гри.

Тож рушій забезпечує надзвичайно високу якість графіки, але потребує потужного комп'ютера, здатного впоратися з ресурсомісткістю ефектів [26].

3.3.3 Gotot

Порівняно з двома іншими механізмами, він повністю безкоштовний із відкритим кодом і є частиною Software Freedom Conservancy (благодійної організації, яка підтримує проекти з відкритим кодом).

Особливо відзначається у розробці 2D-ігор завдяки спеціалізованому 2D-рушію, проте доступна й 3D розробка, яка значно поступається попереднім рушіям. Продуктивність у цьому рушії є пріоритетною, тому він має легку вагу та швидко завантажується [28].

Використовує власну мову скриптингу, а також підтримує C++ та C#.

Таблиця 3.1 – Порівняння ігрових рушіїв

Характеристика	Unity	Unreal Engine	Gotot
1	2	3	4
Інтуїтивно зрозумілий інтерфейс	+	-	+

Кінець таблиці 3.1

1	2	3	4
Підтримка ком'юніті	+	+-	-
Відкритий код	-	+	+
Підтримка 2D	+	-	+
Підтримка 3D	+	+	+
Візуальний скриптинг	+	+	-
Адаптивність	+	+-	-
Маштабованість	+	+	+
Кросплатформеність	+	+	+-

Провівши аналіз, було зроблена висновок, що для реалізації поточного проєкту найкраще рішенням буде використання рушія Unity. Оскільки він легкий у вивченні, має якісну документація та технології 3D візуалізації також має великий магазин об'єктів, а його гнучкість та продуктивність забезпечать ефективну розробку проєкту будь-якої складності.

Unreal Engine теж підходить для реалізації обраного проєкту, але його використання більш доречне у проєктах заснованих на реалізмі. В даному випадку він буде занадто громістким, а його освоєння займе багато часу.

Gotot не підходить для реалізації проєкту через його слабку підтримку 3D розробки.

3.4 Вибір мови програмування

Мова програмування буде безпосередньо залежати від обраного ігрового рушія. Проте розглянувши детально різниці між мовами обраних рушіїв можна сформулювати чіткіше поняття про самі системи та те, наскільки кожна з них підходить для обраної задачі.

C++ та C# мають схожий синтаксис, проте задовольняють різні вимоги. Ці мови мають схожі ознаки такі як поліморфізм, приховування даних, інкапсуляція та інше [29]–[30].

C++ має ручне керування пам'яттю, що дає високу продуктивність, проте може призводити до втрати пам'яті. Швидкість виконання є дуже високою, не зважаючи на довгу компіляцію. Також мінусом є стрімка крива навчання.

C# на відміну від C++ має автоматичний сміттєзбирач, що мінімізує помилки у роботі з пам'яттю. Він має швидшу компіляцію та непогану продуктивність, хоч і потребує більше пам'яті для роботи. Має простий синтаксис та нижчий поріг входу [29]–[30].

Тож C# є чудовим вибором для невеликих чи інди ігор. Тоді як C++ більше підходить для масштабних високоякісних 3D ігор.

3.5 Вибір середовища програмування

На етапі розробки важливим кроком є вибір зручного та функціонального середовища розробки. Основним критерієм відбору є підтримка програмування мовою C#, підтримка плагінів та багатовіконного інтерфейсу. Для аналізу обрано два найпоширеніші середовища для розробки на Unity : Microsoft Visual Studio та JetBrains Rider.

3.5.1 Microsoft Visual Studio

Visual Studio – інтегроване середовище розробки, розроблене компанією Microsoft. Одне з найпоширеніших середовищ розробки для C#. Надає повний набір інструментів для створення програмного забезпечення різної складності. Має потужний функціонал та офіційну підтримку, що забезпечує глибоку інтеграцію з Unity, включаючи IntelliSense, налагодження

коду, підсвічування синтаксису та перегляду стеків [31]. Тому середовище є базовим для багатьох розробників.

Переваги:

- повна сумісність з Unity;
- безкоштовна версія;
- потужний відлагоджувач;
- широкий вибір розширень.

Недоліком є потреба у значних об'ємах пам'яті.

3.5.2 JetBrains Rider

JetBrains Rider – кросплатформне середовище розробки, створене для .NET та C#. Поєднує рушій коду IntelliJ та ReSharper – плагіном для глибокого аналізу коду. Така особливість робить його привабливим для досвідчених розробників. Має офіційну підтримку Unity через додатковий плагін, що дозволяє глибоку інтеграцію з ігровим рушієм. IDE розуміє Unity-сцени, скриптові прив'язки, об'єкти типу GameObject та компоненти [32].

Переваги:

- висока продуктивність;
- автоматичне завершення та підказки коду;
- зручна інтеграція з інструментами;
- глибоке розуміння Unity API;
- інтеграція з Unity Hub;
- можливість персоналізації інтерфейсу.

Програма має платну ліцензію та потребує попередніх налаштувань перед запуском [32].

У таблиці 3.2 представлено порівняння середовищ розробки за ключовими критеріями.

Таблиця 3.2 – Порівняння середовищ розробки

Характеристика	Microsoft Visual Studio	JetBrains Rider
Інтеграція з Unity	+	+
Можливість роботи на різних платформах	+	+
Безкоштовна ліцензія	+	-
Якісна документація	+	+
Оптимізованість	-	+
Система контролю версій	+	+
Підказки/автозавершення	+	+
Швидкий старт	+	-
Приємний дизайн	+	+

3.6 Висновки за розділом 3

В даному розділі розглянуто було проведено аналіз та обґрунтовано вибори основних технічних . Визначено функціональні вимоги для продукту, які відповідають за зручність гри, її привабливість та гарний користувацький досвід.

Було порівняно популярні ігрові рушії – Unity, Unreal Engine та Godot. У результаті порівняння було обрано Unity, як оптимальний варіант для реалізації гри за його базу знань, гнучкість, продуктивність та зручний візуальний редактор.

У якості мови програмування виступила C#, що є основною мовою у Unity. Середовищем розробки було вибрано Microsoft Visual Studio тому, що вона забезпечує інтеграцію з Unity, має потужні засоби відлагодження та підтримує IntelliSense.

4 КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розробка ігрового процесу

Ігровий процес – це спосіб взаємодії гравця з ігровим продуктом [32]. Він формує користувацький досвід, визначаючи виклики, дії та відгуки системи. Ігровий цикл в свою чергу є повторюваною послідовністю дій гравця, що формує ігровий процес.

Основою процесу відеоігор є повторюваний цикл дія-реакція-зворотній зв'язок [33]. Такий

4.1.1 Дія

Дія – це ініціатива гравця, спрямована на зміну стану середовища. У грі це реалізується шляхом керування рухом персонажа за допомогою клавіш управління рухом. Гравець може взаємодіяти з монетами збираючи їх. Також гра потребує взаємодії з кінцевими воротами для завершення гри. Додатково гравець може керувати швидкістю персонажа використовуючи shift [34].

4.1.2 Реакція

Реакція – це зміна ігрового середовища у відповідь на дії гравця. Реакція може бути миттєвою або відкладеною. Головними їх критеріями є логічність та обґрунтованість.

Гра рахує кількість зібраних монет та кількість часу, яка пройшла з моменту початку гри. Під час натиску клавіш руху персонаж переміщається по карті, а під час натиску shift змінює швидкість героя [34].

4.1.3 Відгук

Відгук – форма зворотного зв’язку, яка інформує гравця про результати його дій. Відгуки бувають різних типів візуальні (анімації), звукові (ефекти), тактильні (вібрації) чи ігрові (зміна кількості здоров’я, кількості ресурсів, тощо). Отриманий відгук надає гравцю інформацію, яка допомагає у прийнятті подальших рішень [34].

Зверху на екрані відображається таймер, який надає інформацію про кількість часу з моменту початку гри. Після закінчення гри відображається поточний час проходження лабіринту та найкращий. Також відображається кількість зібраних монет, які підраховуються під час гри та загальної кількості монет на карті. На панелі виграшу наявні кнопки для завершення гри та повторного проходження.

4.2 Структура проєкту

Діаграма класів (рисунок 4.1) відображає структура основних компонентів гри та логіку їх взаємодії. Класи розділені за їх функціональним призначенням. Класи «менеджери» виступають у ролі моделей, зберігаючи та оброблюючи дані. Класи типу Trigger виступають у ролі контролерів реагуючи на дії гравця. А Pausa, Menu та GameEndPanel виконують роль представлення демонструючи користувачу результати гри та меню управління.

Загальна архітектура проєкту структурована для підтримання основних принципів модульності, простоти налаштування та повторного використання коду. Створена діаграма дозволяє виділити основні вузли та чітко поділити їх на функціональні частини.

Проєкт складається з багатьох елементів: скриптів, які забезпечують основну логіку програми; графічних моделей та візуальних ефектів; графічного інтерфейсу; а також звукового супроводу.

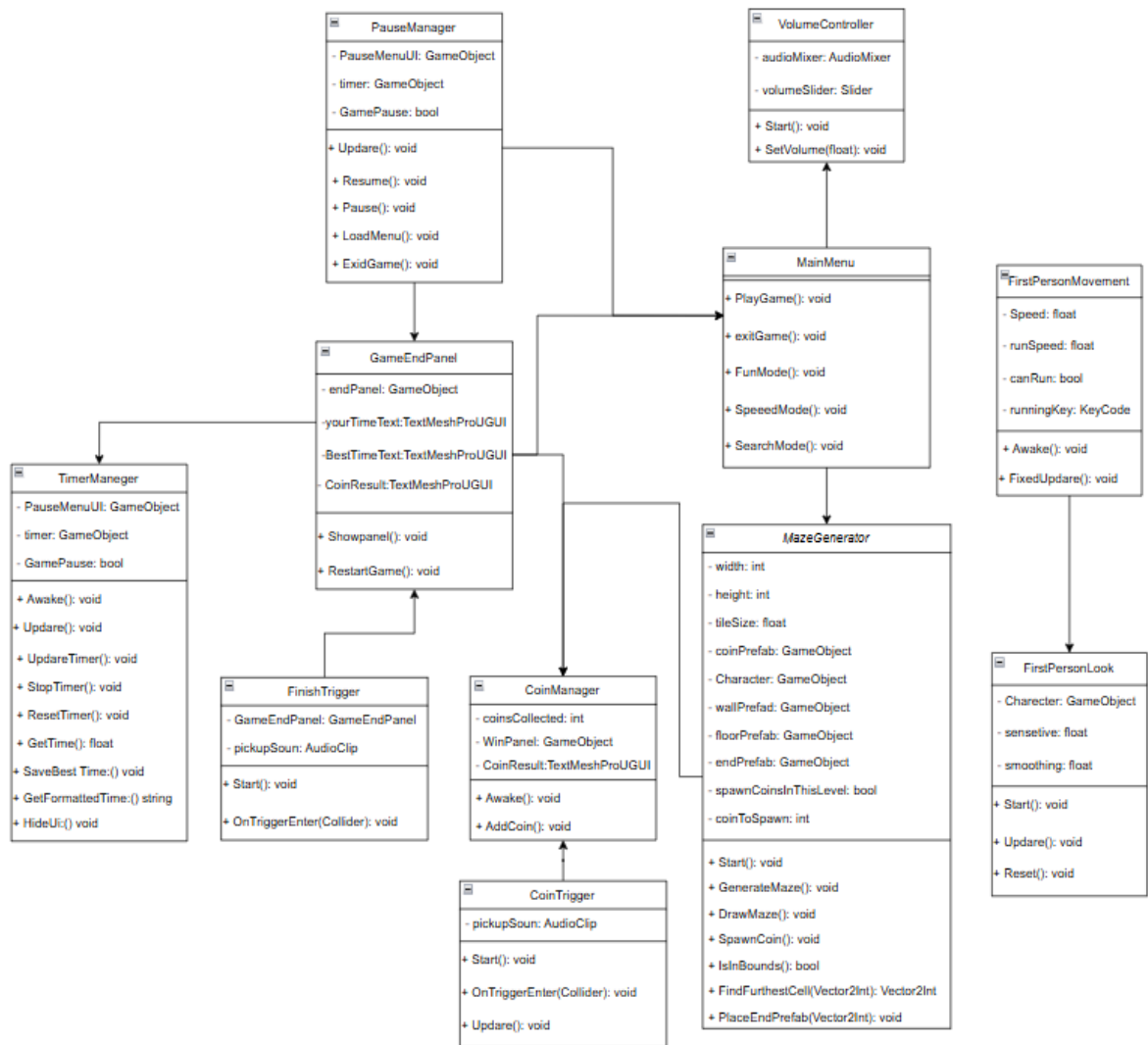


Рисунок 4.1 – Діаграма класів

Логіка програми побудована незалежно від візуального інтерфейсу, що дозволяє повністю модернізувати користувацький інтерфейс програми без проблем з адаптацією логіки програми. У таблиці 4.1 наведено опис всіх скриптів, що використовуються у програмі.

Таблиця 4.1 – Опис файлів

Назва файлу	Опис
1	2
CoinManeger	Відповідає за підрахунок монет та кількість монет у грі

Кінець таблиці 4.1

1	2
CoinTrigger	Обробляє зіткнення гравця з монетою; запускає звуковий ефект; знищує об'єкт та відповідає за його обертання
FinishTrigger	Фіксує досягнення гравцем кінцевої точки гри та активує виграшну панель
MainMenu	Керує логікою головного меню
MapGeneratio	Відповідає за генерацію лабіринту, розміщення об'єктів
PauseMenu	Забезпечує взаємодію з меню паузи та керує зупинкою та відтворенням гри
TimeManager	Відстежує та оновлює таймер гри, зберігає найкращий результат
VolumeController	Забезпечує керування гучність гри
GameEndPanel	Відповідає за відображення виграшної панелі

4.3Опис особливостей програмної реалізації

У рамках розробленої гри розроблено клас MapGeneratio, який відповідає за створення ігрового середовища на основі алгоритму процедурної генерації. Код класу надано у додатку б. У таблиці 4.2 наведено короткий опис призначення кожного з методів, що забезпечують логіку побудови та взаємодії з ігровим світом.

Таблиця 4.2 - Опис функцій класу MapGeneratio

Назва функції	Опис
1	2
Start()	Головна точка запуску. Викликає генерацію лабіринту, його візуалізацію та, за потреби спавн монет.

Кінець таблиці 4.2

1	2
GenerateMaze()	Реалізує алгоритм Recursive Backtracker.
IsInBounds(pos)	Перевіряє чи входить позиція у межі лабіринту. Використовується для уникнення помилок виходу за межі.
FindFurthestCell(from)	Знаходить найвіддаленішу клітинку від заданої за допомогою BFS. Використовується для спавну фінішу.
DrawMaze()	Виводить сітку алгоритму на сцену: створює стіни, підлогу, гравця.
PlaceEndPrefab(exitcell)	Розміщує кінцевий об'єкт(фініш) у найвіддаленішій точці.
SpawnCoin()	Якщо активний відповідний режим, розміщує монети в випадкових прохідних клітинках.

4.4 Опис реалізації графічного інтерфейсу

У проєкті було реалізовано графічний інтерфейс користувача, який забезпечує виведення ключової інформації та взаємодію гравця з нею. Інтерфейс створено з використанням вбудованих засобів Unity. Основними елементами є:

- головне меню — яке дає змогу користувачу розпочати гру та вибрати потрібний режим чи налагодити гру;
- панель паузи — активується при натисненні Escape. Дозволяє зупинити гру та повернутися в головне меню;
- панель завершення гри — з'являється після досягнення кінцевої точки. Містить інформацію про результати гри в залежності від обраного режиму. Має кнопки повернення та початку гри знову;

– додаткові елементи: таймер, лічильник монет та панель налаштування. Є допоміжними частинами для покращення користувацького досвіду.

Усі панелі згруповані в один Canvas та розміщені відповідно до потреб геймплею.

4.5 Висновки за розділом 4

У розділі було вирішено завдання конструювання програмного забезпечення. Було досліджено ігровий процес та його етапи. Що дає змогу ефективно розділити процес на етапи для подальшої розробки.

Створено структуру проєкту, яка складається з окремих функціональних модулів: генерація лабіринту, керування гравцем, обробка колізій, управління гнучкістю та графічним інтерфейсом.

Розроблено алгоритм процедурної генерації лабіринту з використанням Recursive Backtracker, а також реалізовано пошук найвіддаленішої точки методом пошуку в ширину. Це забезпечує неповторність ігрового середовища при кожному запуску гри.

Також реалізовано графічний інтерфейс користувача, що включає головне меню, панель паузи, панель завершення гри та панель налаштувань.

Таким чином у межах розділу було реалізовано ключові механіки гри, структуру проєкту та ігрову логіку для забезпечення повноцінної взаємодії гравця з ігровим світом.

5 ІНСТРУКЦІЯ З ВИКОРИСТАННЯ ПРОГРАМНОГО ПРОДУКТУ

В розділі описується інструкція з використання програми. Нижче описані системні вимоги, інструкція з запуску та опис роботи системи.

5.1 Системні вимоги

Для коректної роботи програмного продукту необхідні такі апаратні вимоги:

- операційна система: Windows 7/8/9/10/11;
- процесор з частотою 2.2 ГГц;
- відеокартка: інтегрована (з підтримкою OpenGL 3.3 та вище);
- вільне місце на диску: 300мб;
- операційна пам'ять: не менше 2 ГБ.

5.2 Функціонал системи

Програмний продукт реалізує наступні можливості:

- генерація ігрової карти;
- взаємодія з об'єктами ігрового світу;
- візуальні та звукові ефекти;
- підрахунок часу проходження рівня;
- керування гравцем за допомогою клавіш W, A, S, D.

5.3 Інструкція з запуску системи

Для запуску програми спочатку треба завантажити архів з програмою. Архів треба розпакувати у зручному місці з достатньою кількістю пам'яті. Після чого запустити exe-файл гри.

5.4 Опис роботи програми

Для виконання програми треба запустити її через exe-файл гри. На рис. 5.1 зображене головне меню гри.



Рисунок 5.1 – Головне меню гри

Для зміни гучності треба натиснути на кнопку «Setting» та відкриється вікно налаштувань (рис. 5.2).

Для регуляції загального звуку гри використовується слайдер.

Перед початком гри користувач має вибрати один з трьох запропонованих режимів гри (рис. 5.3), а саме :

- звичайний режим;
- режим на швидкість;
- режим зі збором об'єктів.

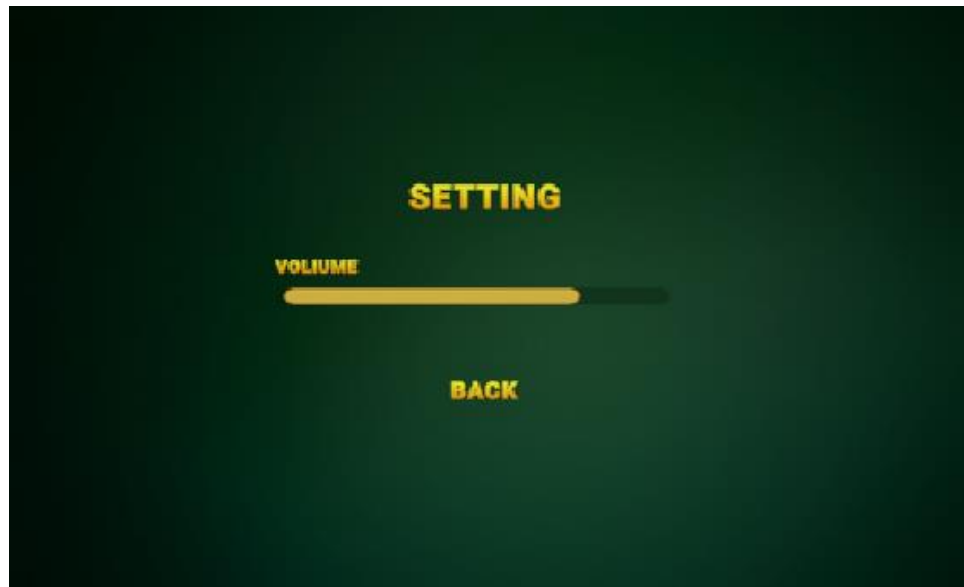


Рисунок 5.2 – Панель налаштувань гри

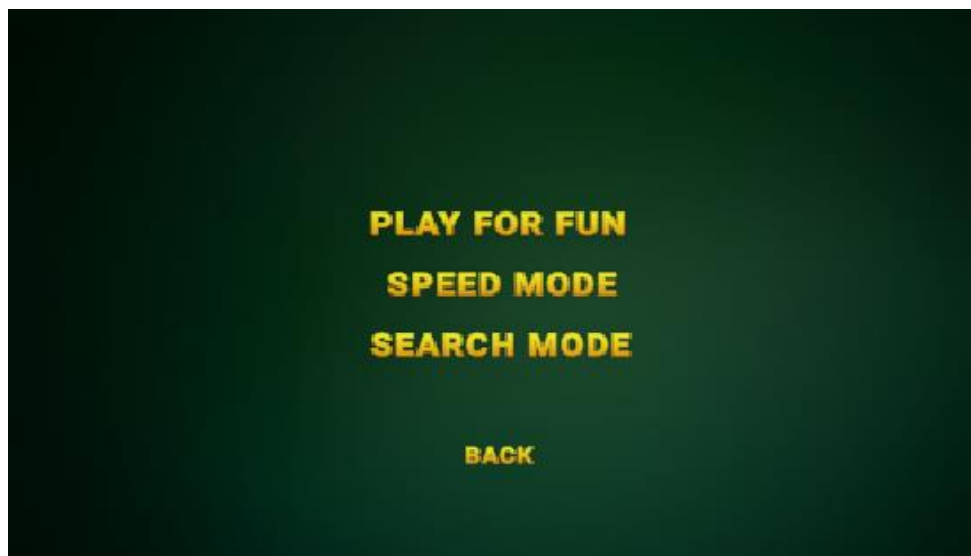


Рисунок 5.3 – Режими гри

Для завершення гри гравець має віднайти арку, яка випадковим чином розміщений на карті (рис. 5.4). Після чого з'явиться вікно завершення гри. Інформація, яка подана на вікнах відрізняється в залежності від обраного режиму. У звичайному режимі користувач має лише одне завдання – дістатися до фінішу. У вікні завершення цього режиму (рис. 5.5) відображається лише повідомлення про успішне завершення гри. Також присутні кнопки виходу в меню та повтору гри.



Рисунок 5.4 – Арка

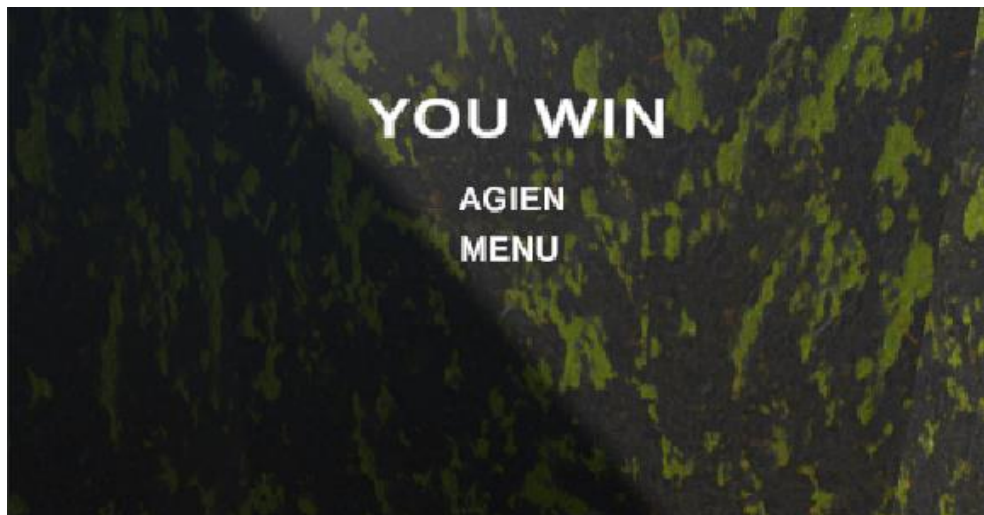


Рисунок 5.5 – Панель звичайного режиму

У режимі на швидкість на екран додатково виводиться таймер, який фіксує час проходження рівня. Після завершення гри користувачу надається інформація про його результат та найкращий час. У режимі зі збором об'єктів на карті випадковим чином розміщуються монети. Завдання гравця зібрати якомога більше монет до завершення рівня. Гравець не знає точної кількості монет на карті, вона вибирається випадковим чином у заданому діапазоні. При закінченні гри гравцю показують кількість зібраних монет (рис. 5.7) від загальної кількості (рис. 5.6).



Рисунок 5.6 – Панель пошукового режиму

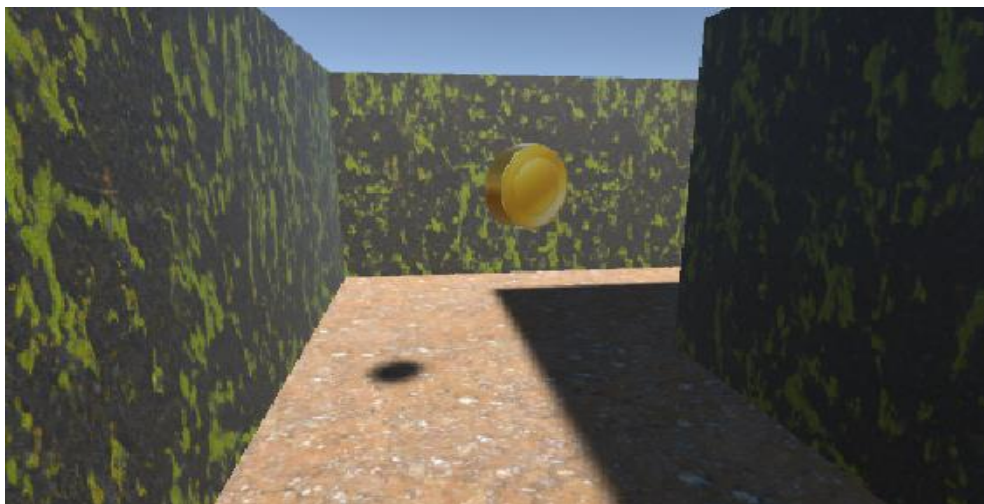


Рисунок 5.7 – Монета

5.5 Висновки до розділу 4

У цьому розділі була розроблена інструкція з використання програмного продукту. Це знайомить користувача з основними правилами гри та формує базове розуміння взаємодії з ігровим середовищем.

Були описані системні вимоги для коректної роботи та функціонування гри. Детально описана інструкція запуску системи. Наведено основні елементи програми та їх можливості.

ВИСНОВКИ

В результаті виконання дипломної роботи було проведено повний цикл дослідження, розробки та реалізації комп'ютерної гри з динамічною генерацією мапи. Метою роботи стало створення функціональної гри, яка б реалізувала механіку процедурного створення ігрового середовища, що змінюється з кожним запуском гри.

У теоретичній частині було проведено аналіз сучасного стану ігрової індустрії, розглянуто особливості та етапи розробки комп'ютерних ігор, їх класифікацію та актуальні тенденції. Було досліджено сучасні аналоги ігор з використанням процедурної генерації. Виділено їх особливості та ефективності процедурної генерації для створення ігор без ручного проєктування рівнів.

Було досліджено класифікації та різновиди лабіринтів для розуміння загальної картини задачі. Досліджено популярні алгоритми генерації – такі як бінарне дерево, алгоритм Прима, Recursive Backtracker та Hunt-and-kill. На основі аналізу було обрано найоптимальніший варіант для реалізації.

У практичній частині було обґрунтовано вибір рушія Unity, як основної технологічної платформи, яка забезпечує гнучкість у створенні 3D-середовищ. Реалізація гри передбачала створення алгоритму генерації лабіринту та додаткових об'єктів, модулі керування гравцем, обробки колізій та створення користувацького інтерфейсу.

Результатом роботи є функціональний ігровий застосунок, який демонструє базові принципи роботи процедурної генерації, реалізацію UI, роботу з тривимірним простором та обробку подій.

Логіка програми написана з врахуванням можливості подальшого масштабування та розширення. Для розширення проєкту можна покращити графічні елементи, додати ворогів та зброю для бою. Також можна розширити кількість режимів гри, наприклад з обмеженням в часі чи подоланням

перешкод. Для ускладнення шляху можна модернізувати основний лабіринт з додаванням петель. Цікавим рішенням може бути створення потаємних проходів.

Проект демонструє актуальність використання алгоритмів генерації ігрових світів, важливість продуманого інтерфейсу користувача та доцільність поєднання технічних та креативних підходів у створенні ігрових застосунків.

Результати роботи було представлено на Щорічній науково-практичній конференції викладачів, науковців, молодих учених, аспірантів та студентів Національного університету «Запорізька політехніка» – «Тиждень науки-2025».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. The Past, Present, and Future of Gaming [Electronic resource]. – Access mode: <https://blog.purestorage.com/perspectives/from-arcades-to-metaverse-the-past-present-and-future-of-gaming/>.
2. How the Video Game Industry Is Changing [Electronic resource]. – Access mode: <https://www.investopedia.com/articles/investing/053115/how-video-game-industry-changing.asp#toc-vintage-video-games>.
3. Adams, T. Procedural Generation in Game Design / T. Adams, T. Short. – CRC Press, 2017 – 336 p.
4. The Social Impact of Games in Modern [Electronic resource]. – Access mode: <https://www.sae.edu/gbr/insights/the-social-impact-of-games-in-modern-society/>.
5. Як розробляють ігри? [Електронний ресурс] – Режим доступу: <https://lemon.school/blog/yak-rozroblyayut-igry>.
6. The Seven Stages Of Game Development [Electronic resource]. – Access mode: <https://gamemaker.io/en/blog/stages-of-game-development>.
7. The 7 Stages of the Game Development Process [Electronic resource]. – Access mode: <https://stepico.com/blog/game-development-stages/>.
8. Video Game Genres: Everything You Need to Know [Electronic resource]. – Access mode: <https://www.hp.com/us-en/shop/tech-takes/video-game-genres>.
9. Нікітін, С.О Основи комп'ютерних ігор та ігрових програм / С. О. Нікітін. – Х. : «Друкарня Мадрид», 2018. – 138 с.
10. Мудрий, Я. Г. Комп'ютерні ігри та їх класифікація / Я.Г. Мудрий // Актуальні питання сучасної інформатики. – 2016. – №2. – С. 133-138.
11. 10 Best Procedurally-Generated Games [Electronic resource]. – Access mode: <https://www.thegamer.com/best-procedurally-generated-games/>.
12. Yu, D. Spelunky / D. Yu // – Boss First Book, 2016. – 209 p.

13. Below Decks at Ghost Ship: Cave generation in Deep Rock Galactic [Electronic resource]. – Access mode: <https://steamcommunity.com/games/DeepRockGalactic/announcements/detail/4593196713081471259>.
14. Enter the Gungeon [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Enter_the_Gungeon.
15. Dungeon Generation in Enter The Gungeon [Electronic resource]. – Access mode: <https://www.boristhebrave.com/2019/07/28/dungeon-generation-in-enter-the-gungeon/>.
16. Invisible, Inc [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Invisible,_Inc.
17. Maze [Electronic resource]. – Access mode: <https://en.wikipedia.org/wiki/Maze>.
18. Maze Classification [Electronic resource]. – Access mode: <https://www.astrolog.org/labyrinth/algrithm.htm>.
19. Maze Generation Algorithms - An Exploration [Electronic resource]. – Access mode: <https://professor-l.github.io/mazes/>.
20. Аналіз та порівняння існуючих методів генерації лабіринтів в комп'ютерних іграх / [Г. В. Марчук, М. С. Граф, В. Л. Левківський, та інші] // Вісник Херсонського національного технічного університету. – 2024. - №3. – С. 128-137.
21. Watkins, R. Procedural Content Generation for Unity Game Development / R. Watkins // - Packt Publishinh, 2016. – 260 p.
22. Singleton [Electronic resource]. – Access mode: <https://refactoring.guru/design-patterns/singleton>.
23. Unity Documentation. Introduction to prefabs [Electronic resource]. – Access mode: <https://docs.unity3d.com/6000.1/Documentation/Manual/prefabs-introduction.html>.

24. Ігровий рушій [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%86%D0%B3%D1%80%D0%BE%D0%B2%D0%B8%D0%B9_%D1%80%D1%83%D1%88%D1%96%D0%B9.
25. Unity [Electronic resource]. – Access mode: <https://unity.com/>.
26. Створення ігор: вибір ігрового рушія [Електронний ресурс] – Режим доступу: <https://cyberglads.com>.
27. Unreal Engine 4 [Electronic resource]. – Access mode: <https://www.unrealengine.com/en-US/>.
28. Godot [Electronic resource]. – Access mode: <https://godotengine.org/>.
29. C# vs. C++: What's at the Core? [Electronic resource]. – Access mode: <https://www.toptal.com/c-sharp/c-sharp-vs-c-plus-plus>.
30. C++ [Electronic resource]. – Access mode: <https://en.wikipedia.org/wiki/C%2B%2B>.
31. Visual Studio [Electronic resource]. – Access mode: <https://visualstudio.microsoft.com>.
32. Rider JetBrains [Electronic resource]. – Access mode: <https://www.jetbrains.com/rider/>.
33. Gameplay [Electronic resource]. – Access mode: <https://en.wikipedia.org/wiki/Gameplay>.
34. How to design gameplay loops that keep players coming back for more [Electronic resource]. – Access mode: <https://supersonic.com/learn/blog/how-to-design-gameplay-loops-that-keep-players-coming-back-for-more/>.
35. Maze Generation: Recursive Backtracking [Electronic resource]. – Access mode: <https://weblog.jamisbuck.org/2010/12/27/maze-generation-recursive-backtracking>.

ДОДАТОК А
Технічне завдання

Вступ

Мета проєкту - програмна реалізація алгоритму побудови лабіринту для гри з механікою динамічної генерації мапи.

Опис поточного стану. На ринку присутні ігри з динамічною генерацією в різних напрямках. Серед класичних ігор відсутня реалізація динамічної генерації, що обмежує їх можливості.

Опис бажаного стану. Створити на ринку модернізовану версію класичної гри, яка б реалізувала її потенціал. Відновити популярність гри у фанатів та залучити нових користувачів.

A.1 Обсяг робіт

A.1.1 Вимоги до інтеграції

Програма є локальною та має такі вимоги до внутрішньої інтеграції:

- збереження результатів гри в локальному файлі;
- читання даних конфігурації.

A.1.2 Функціональні вимоги

Операції які мають виконуватися:

- ініціалізація та передача даних при генерації лабіринту;
- збереження результатів на пристрої у вигляді файлів;
- завантаження конфігурації при запуску гри.

A.1.3 Технічні вимоги

Для експлуатації локальної гри потрібне таке апаратне та програмне забезпечення:

- не менше 2 Гб вільної пам'яті на пристрої;
- процесор з частотою 2.2 ГГц;
- не менше 2000 Мб операційної пам'яті;
- відеоадаптер з підтримкою OpenGL 3.3 та вище;
- 64-bit операційна система.

A.2 Виконання робіт

Розробка включає в себе сім основних етапів: планування, попереднє виробництво, виробництво, тестування, передзапуск, запуск та супровід.

На першому етапі буде проведено аналіз ринку та цільової аудиторії. Також на цьому етапі буде розплановано ресурси проєкту та приблизні терміни виконання.

На другому етапі буде оформлено загальний концепт гри. Буде створена документація. Виконання загальної підготовки, що включає вибір механік та геймплєю, нариси персонажів та інше, допоможуть у створенні документації.

На етапі виробництва буде створена програмне реалізація, додана графіка та музика.

Етап тестування це процес для виявлення неполадок у грі. Будуть проведені функціональні тестування.

На момент передзапуску гра буде мати стабільну версію. На цьому етапі будуть усунені всі помилки виявлені при тестуванні.

На етапі запуску гра буде розміщена у ігрових магазинах.

Етап постпродакшену буде зосереджений на доповненні гри та виправленні нових помилок.

A.3 Тестування

Метою тестування є перевірити коректність роботи всіх основних функцій програми. Буде проведено функціональне та ітераційне тестування. Перевірено функції генерації лабіринтів, збереження даних, коректного руху гравця, обробка зіткнень завершення гри та інше.

A.4 Критерії приймання

Тестування вважатиметься виконаним при таких умовах:

- усі функції працюють без помилок;
- дані з лабіринту коректно передаються в логіку гри;
- UI коректно оновлюється;
- гра коректно зберігає та завантажує дані;
- гра відкривається на цільових системах.

A.5 Документація

Для даного проєкту повинна бути створена технічна документація. Обов'язково має бути створений звіт з тестування. Опціонально можуть бути створені інструкції для користувачів.

A.6 Безпека

Програма буде розміщена локально на персональних комп'ютерах, тож головними вимогами до безпеки програми є шифрування файлів та блокування доступу до внутрішніх інструментів. Це забезпечить захист персональних даних гравця. Також подібні заходи безпеки будуть запобігати зламам та іншим зовнішнім втручанням у гру.

ДОДАТОК Б
Текст програми

Б.1 Файл MapGeneration.cs

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

public class MazeGenerator : MonoBehaviour
{
    public int width = 21;
    public int height = 21;

    public GameObject wallPrefab;
    public GameObject floorPrefab;
    public int tileSize = 1;
    public GameObject startPrefab;
    public GameObject endPrefab;
    public GameObject Character;
    private Vector2Int startPos;
    private Vector2Int endPos;
    public bool spawnCoinsInThisLevel = false;
    public GameObject coinPrefab;
    public int coinsToSpawn;

    private int[,] grid;
    private System.Random rng = new System.Random();
```

```
private Vector2Int[] directions = {
    new Vector2Int(0, 2), // вгору
    new Vector2Int(2, 0), // право
    new Vector2Int(0, -2), // низ
    new Vector2Int(-2, 0) // ліво
};
```

```
void Start()
{
    GenerateMaze();
    DrawMaze();
    if (spawnCoinsInThisLevel)
    {
        SpawnCoins();
    }
}
```

```
void GenerateMaze()
{

    // Створюємо сітку (0 = стіна, 1 = прохід)
    grid = new int[width, height];
    // Стек позицій (для backtracking)
    Stack<Vector2Int> stack = new Stack<Vector2Int>();

    Vector2Int start = new Vector2Int(1, 1);
    grid[start.x, start.y] = 1;
    startPos = start;
```

```

stack.Push(start);

while (stack.Count > 0)
{
    Vector2Int current = stack.Peek();

    List<Vector2Int> unvisitedNeighbors = new List<Vector2Int>();

    foreach (var dir in directions)
    {
        Vector2Int neighbor = current + dir;
        if (IsInBounds(neighbor) && grid[neighbor.x, neighbor.y] == 0)
        {
            Vector2Int between = current + dir / 2;
            if (grid[between.x, between.y] == 0)
                unvisitedNeighbors.Add(dir);
        }
    }

    if (unvisitedNeighbors.Count > 0)
    {
        Vector2Int chosenDir =
unvisitedNeighbors[rng.Next(unvisitedNeighbors.Count)];

        Vector2Int between = current + chosenDir / 2;
        Vector2Int next = current + chosenDir;

        grid[between.x, between.y] = 1;
        grid[next.x, next.y] = 1;
    }
}

```

```

        stack.Push(next);
    }
    else
    {
        stack.Pop();
    }
}
endPos = FindFurthestCell(start);
}

bool IsInBounds(Vector2Int pos)
{
    return pos.x > 0 && pos.y > 0 && pos.x < width - 1 && pos.y < height - 1;
}

Vector2Int FindFurthestCell(Vector2Int from)
{
    Queue<Vector2Int> queue = new Queue<Vector2Int>();
    bool[,] visited = new bool[width, height];
    int[,] distance = new int[width, height];
    List<Vector2Int> candidates = new List<Vector2Int>();
    int maxDist = 0;

    queue.Enqueue(from);
    visited[from.x, from.y] = true;
    distance[from.x, from.y] = 0;

    while (queue.Count > 0)

```

```

{
    Vector2Int current = queue.Dequeue();

    foreach (var dir in directions)
    {
        Vector2Int next = current + dir;
        if (IsInBounds(next) && !visited[next.x, next.y] && grid[next.x, next.y]
== 1)
        {
            visited[next.x, next.y] = true;
            distance[next.x, next.y] = distance[current.x, current.y] + 1;
            queue.Enqueue(next);

            if (distance[next.x, next.y] > maxDist)
            {
                maxDist = distance[next.x, next.y];
                candidates.Clear();
                candidates.Add(next);
            }
            else if (distance[next.x, next.y] == maxDist)
            {
                candidates.Add(next);
            }
        }
    }

    if (candidates.Count > 0)
    {

```

```

    Vector2Int best = candidates[0];
    int bestEdgeDist = Mathf.Min(best.x, best.y, width - 1 - best.x, height - 1 -
best.y);

    foreach (var cell in candidates)
    {
        int edgeDist = Mathf.Min(cell.x, cell.y, width - 1 - cell.x, height - 1 -
cell.y);
        if (edgeDist < bestEdgeDist)
        {
            best = cell;
            bestEdgeDist = edgeDist;
        }
    }
    return best;
}

return from;
}

```

```

void DrawMaze()
{
    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {

```

```

    Vector3 pos = new Vector3(x * tileSize, 0, y * tileSize);
    if (grid[x, y] == 0)
    {
        Instantiate(wallPrefab, pos, Quaternion.identity);
    }
    else
    {
        Instantiate(floorPrefab, pos, Quaternion.identity);
    }
}

Instantiate(startPrefab, new Vector3(startPos.x * tileSize, 0.5f, startPos.y *
tileSize), Quaternion.identity);

Instantiate(Character, new Vector3(startPos.x * tileSize, 0.5f, startPos.y *
tileSize), Quaternion.identity);

PlaceEndPrefab(endPos);
}

void PlaceEndPrefab(Vector2Int exitCell)
{
    Vector3 worldPos = new Vector3(exitCell.x * tileSize, 0, exitCell.y * tileSize);
    Quaternion rotation = Quaternion.identity;

    //перевірка проходу
    foreach (Vector2Int dir in directions)
    {
        Vector2Int neighbor = exitCell + dir;
        Debug.Log($"Checking neighbor {neighbor}");
    }
}

```

```

    if (IsInBounds(neighbor) && grid[neighbor.x, neighbor.y] == 1)
    {
        Debug.Log($"Found passage at direction: {dir}");

        //обертання воріт
        if (dir == new Vector2Int(0, 2)) rotation = Quaternion.Euler(0, 0, 0); //
вгору
        else if (dir == new Vector2Int(2, 0)) rotation = Quaternion.Euler(0, 90, 0);
// право
        else if (dir == new Vector2Int(0, -2)) rotation = Quaternion.Euler(0, 180,
0); // низ
        else if (dir == new Vector2Int(-2, 0)) rotation = Quaternion.Euler(0, 270,
0); // ліво
        break;
    }
}

Debug.Log(rotation);
GameObject empty = new GameObject("EndWrapper");
empty.transform.position = worldPos;
empty.transform.rotation = rotation;

GameObject end = Instantiate(endPrefab, worldPos, Quaternion.identity);
end.transform.SetParent(empty.transform);
}

void SpawnCoins()
{

```

```

CoinManager.Instance.totalCoins = coinsToSpawn;
int spawned = 0;
while (spawned < coinsToSpawn)
{
    int x = Random.Range(1, width - 1);
    int y = Random.Range(1, height - 1);

    if (grid[x, y] == 1)
    {
        Vector3 pos = new Vector3(x * tileSize, 0.5f, y * tileSize);
        Instantiate(coinPrefab, pos, Quaternion.identity);
        spawned++;
    }
}
}

```

Б.2 Файл PauseMenu.cs

```

using System.Threading;
using UnityEngine;
using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour
{

```

```
public static bool GamePaused = false;
public GameObject pauseMenuUI;
public GameObject timer;

void Update()
{
    if(Input.GetKeyDown(KeyCode.Escape))
        if(GamePaused)
        {
            Resume();
        }else
        {
            Pause();
        }
}

public void Resume()
{
    pauseMenuUI.SetActive(false);
    Time.timeScale = 1;
    GamePaused = false;
    Cursor.lockState = CursorLockMode.Locked;
    Cursor.visible = false;
    if(timer != null) { timer.SetActive(true); }
}
```

```
void Pause()
{
    pauseMenuUI.SetActive(true);
    Time.timeScale = 0;
    GamePaused = true;
    Cursor.lockState = CursorLockMode.None;
    Cursor.visible = true;
    if (timer != null) { timer.SetActive(false); }
}

public void LoadMenu()
{
    Time.timeScale = 1;
    SceneManager.LoadScene(0);
}

public void ExitGame()
{
    Application.Quit();
}

public void RestartLevel()
{
    Scene currentScene = SceneManager.GetActiveScene();
    SceneManager.LoadScene(currentScene.name);
}
}
```

ДОДАТОК В
Слайди презентації

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»
Кафедра програмних засобів

Дипломна кваліфікаційна робота бакалавра на тему:

ПРОГРАМНА РЕАЛІЗАЦІЯ ГРИ З ДИНАМІЧНОЮ ГЕНЕРАЦІЄЮ МАПИ

Виконала
ст. гр КНТ-111

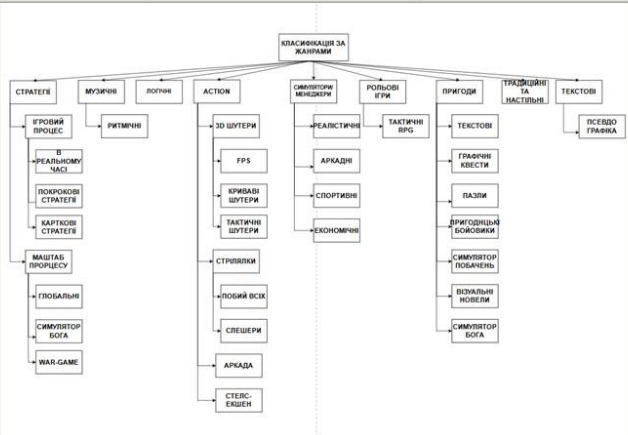
Анастасія БАКЛИЦЬКА

Керівник
к.т.н доцент

Анжеліка ПАРХОМЕНКО

Рисунок В.1 – Титульний слайд

Актуальність роботи



Діаграма жанрів комп'ютерних ігор

Рисунок В.2 – Актуальність роботи

Огляд існуючих аналогів



Spelunky



Invisible, Inc



Enter the Gungeon



Deep Rock Galactic

3

Рисунок В.3 – Огляд існуючих аналогів

Мета та завдання роботи

Метою роботи є програмна реалізація алгоритму побудови лабіринту для гри з механікою динамічної генерації мапи.

Завдання роботи:

- дослідити існуючі алгоритми генерації ігрової мапи та обґрунтувати вибір одного з них для реалізації ігрового застосунку з процедурно згенерованим середовищем;
- виконати проєктування та конструювання програмного забезпечення;
- розробити інструкцію з використання програми.

4

Рисунок В.4 – Мета та завдання роботи

Аналіз функціональних вимог

Програма повинна реалізувати наступні функції
Переміщення персонажа у 3D просторі
Реалізація камери від першої особи
Генерація лабірину
Реалізація механіки виграшу
Розміщення об'єктів у лабіринті
Налаштування гри

5

Рисунок В.5 – Аналіз функціональних вимог

Порівняння алгоритмів

	Kruskal's	Aldous-Broder	Hunt-and-kill	Binary Tree	Recursive Backtracker
Часова складність	$O(E \log E)$	$O(n^2)$	$O(n)$	$O(n)$	$O(n)$
Тип лабірину	Ідеальний	Ідеальний	Ідеальний	Неідеальний	Ідеальний
Тип алгоритму	Мінімальне-остовне дерево	Стохастичний	Гібридний-пошук	Жадібний	Пошук в глибину
Випадковість	Висока	Дуже висока	Середня	Низька	Висока
Складність реалізації	Висока	Низька	Низька	Дуже низька	Середня
Шлях лабірину	Розгалужений	Рівномірний	Звивистий	Прямолінійний прогнозований	Звивистий
Придатний для 3D	Так	З ускладненнями	Так	Так	Так
Створює довгі коридори	Помірно	Ні	Помірно	Ні	Так

6

Рисунок В.6 – Порівняння алгоритмів

Порівняння ігрових рушіїв

	Unity	Unreal Engine	Godot
Інтуїтивно зрозумілий інтерфейс	+	-	+
Підтримка ком'юніті	+	+-	-
Відкритий код	-	+	+
Підтримка 2D	+	-	+
Підтримка 3D	+	+	+
Візуальний скриптинг	+	+	-
Адаптивність	+	+-	-
Маштабованість	+	+	+
Кросплатформеність	+	+	+-

7

Рисунок В.7 – Порівняння ігрових рушіїв

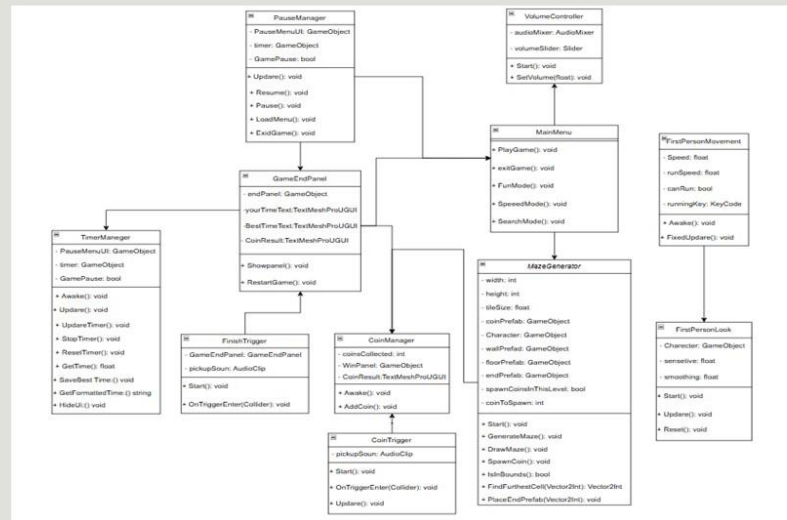
Порівняння середовищ розробки

	Microsoft Visual Studio	JetBrains Rider
Інтеграція з Unity	+	+
Можливість роботи на різних платформах	+	+
Безкоштовна ліцензія	+	-
Якісна документація	+	+
Оптимізованість	-	+
Система контролю версій	+	+
Підказки/автозавершення	+	+
Швидкий старт	+	-
Приємний дизайн	+	+

8

Рисунок В.8 – Порівняння середовищ розробки

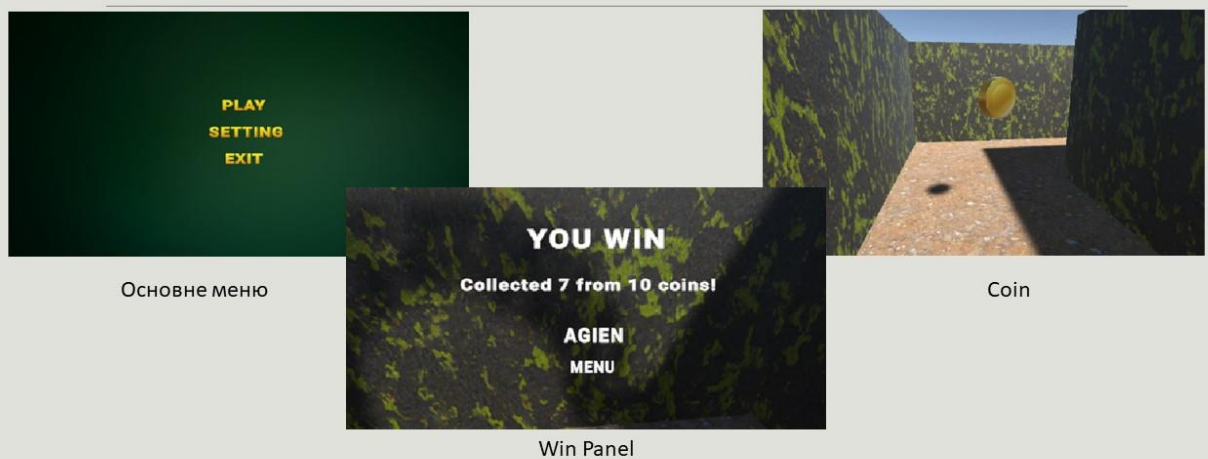
Діаграма класів програми



9

Рисунок В.9 – Діаграма класів програми

Інтерфейс програми



10

Рисунок В.10 – Інтерфейс програми

Приклад згенерованого лабіринту



Варіація лабіринту

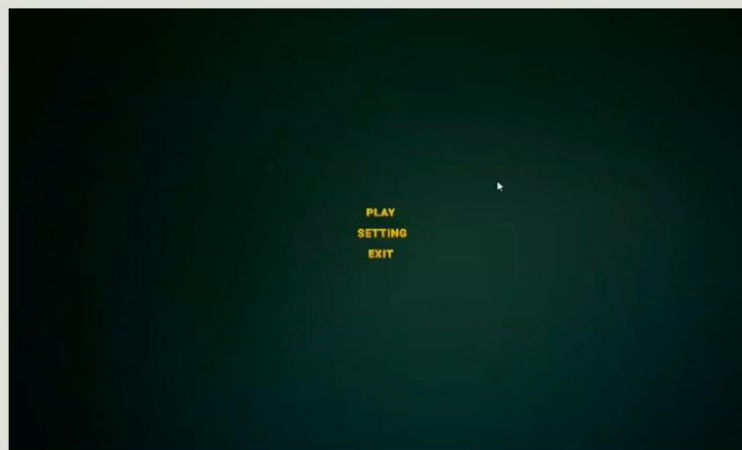


Варіація лабіринту з генерацією монет

11

Рисунок В.11 – Приклад згенерованого лабіринту

Відеодемонстрація гри



12

Рисунок В.12 – Відеодемонстрація гри

Висновки

У результаті виконання дипломної роботи створено функціональну 3D-гру з процедурною генерацією мапи.

Було досліджено стан ігрової індустрії, класифікацію ігор, типи лабіринтів та алгоритми генерації (DFS, Прима, Hunt-and-Kill).

Реалізацію здійснено в середовищі Unity, використано мову C# та середовище розробки Visual Studio.

Гра включає кілька режимів, управління персонажем, взаємодію з об'єктами, таймер і графічний інтерфейс. Програмна архітектура враховує можливість масштабування.

Проект демонструє практичне застосування алгоритмів генерації, сучасних підходів до UI та потенціал подальшого розвитку.

Результати роботи було представлено на Щорічній науково-практичній конференції викладачів, науковців, молодих учених, аспірантів та студентів Національного університету «Запорізька політехніка» – «Тиждень науки-2025».

Рисунок В.13 – Висновки