Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

Розробка roguelite-гри з процедурною генерацією оточення (підземель).

Текстова частина до курсової роботи
за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи
с.в. _Борозенний С.О.__
(прізвище та ініціали)
_____
(підпис)
“____” _____ 2023 р.

Виконав студент _____
___Боровік Н. І.._____
(прізвище та ініціали)
“____” _____ 2023 р.

Міністерство освіти і науки України

Київ 2023

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри мультимедійних систем,
доцент, к.ф-м.н.
_____О. П. Жежерун (підпис)
„____”_____2023 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Боровіку Нікіті Івановичу факультету інформатики 3-го курсу

ТЕМА Розробка roguelite-гри з процедурною генерацією оточення (підземель).

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Вступ
1 Жанр Roguelite та використання процедурної генерації оточення у ньому.
2 Аналіз способів процедурної генерації та опис обраного алгоритму.
3 Розробка
Висновки
Список літератури
Додатки (за необхідністю)

Дата видачі „___” _____ 2023 р. Борозенний О.С. _____ (підпис)
Завдання отримав _____ (підпис)

Тема: Розробка roguelite-гри з процедурною генерацією оточення (підземель).

Календарний план виконання роботи:

№ п/п	Назва етапу курсового проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	01.10.2022	
2.	Огляд та аналіз матеріалів за темою роботи, дослідження існуючих алгоритмів.	01.11.2022	
3.	Проектування комп'ютерної гри.	20.12.2022	
4.	Розробка комп'ютерної гри.	03.04.2023	
5.	Написання текстової частини роботи.	28.04.2023	
6.	Створення доповіді.	10.05.2023	
7.	Захист курсової роботи.	25.05.2023	

Боровік Н. І. _____

Борозенний О. С. _____

"_____"_____

Зміст

Анотація

Курсова робота присвячена розробці комп'ютерної гри в жанрі roguelite, що використовує алгоритм процедурної генерації оточення. В ході розробки комп'ютерної гри було досліджено особливості жанру roguelite та способи процедурної генерації, що використовуються в іграх заданого жанру.

Ключові слова: процедурна генерація, комп'ютерна гра, Unity, roguelite, roguelike.

Вступ

На сьогоднішній день ігрова індустрія займає одне з провідних місць в розвитку інформаційних технологій у нашому світі. Одним із цікавих та популярних напрямів ігрової розробки є створення ігор в жанрі roguelite з процедурною генерацією оточення. Цей жанр набув популярності через свою спроможність забезпечувати гравцю унікальний ігровий досвід під час кожного сеансу гри.

Метою моєї курсової роботи є аналіз способів процедурної генерації оточення, що становить підземелля, в іграх жанру roguelite, а також розробка і впровадження алгоритму процедурної генерації у свою гру.

Текстова частина курсової роботи містить в собі 3 основні розділи:

Перший розділ присвячено жанру roguelite в ігровій розробці та місцю процедурної генерації оточення в ньому.

Другий розділ курсової роботи присвячений аналізу способів процедурної генерації оточення, а також опису обраного для курсової роботи алгоритму процедурної генерації.

Третій розділ презентує засоби, використані в розробці гри, а також в ньому розглянуто класи та методи створеного процедурного генератора.

Основна частина

Розділ 1: Жанр Roguelite та використання процедурної генерації рівнів у ньому.

1.1.Що таке Roguelite гра

Для визначення жанру roguelite слід спочатку згадати його жанр-прабатько – roguelike.

Назва roguelike бере своє походження від назви комп'ютерної гри Rogue, що була випущена в 1980 році та започаткувала жанр roguelike в ігровій розробці. Roguelike[2] – це жанр комп'ютерних ігор, основні критерії якого були визначені 2008 року в Берліні на міжнародній конференції з розробки Roguelike. До основних факторів[1] згідно з Берлінською інтерпретацією були віднесені:

- Генерація випадкового оточення гри: дія гри проходить у процедурно згенерованій послідовності кімнат, що є різною для кожного окремого сеансу гри.
- Остаточна смерть: смерть персонажа має означати повну втрату всього прогресу.
- Вбивство монстрів як основна складова геймплею: битва з ворогами є ключовою складовою геймплею.
- Покроковий рух по сітці з відсутністю реального часу: рух дозволено лише по сітці, а час йде вперед тільки у момент руху чи іншої дії персонажа.
- Складність, обмеженість ресурсів: гра жанру roguelike вимагає розумного використання усіх наданих ресурсів для подолання труднощів.

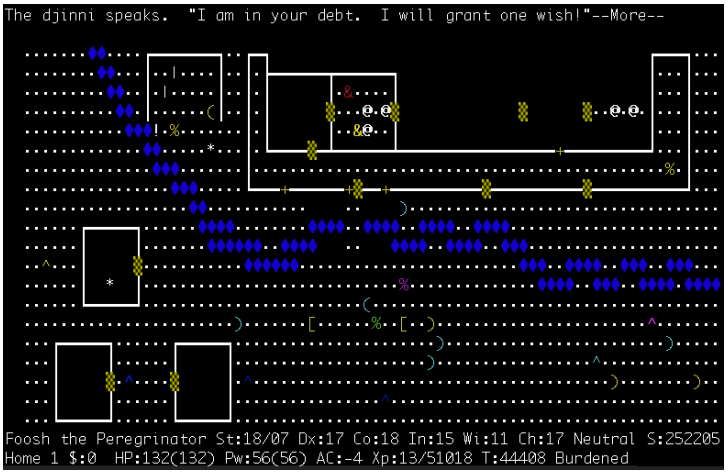

Рис.1 Гра Rogue

Roguelite[2] – жанр комп'ютерних ігор , що походить від roguelike. Ігри в цьому жанрі переймають основні особливості roguelike-ігор (процедурна генерація оточення, складність, битви з ворогами тощо), однак не обов'язково відповідають усім критеріям, визначеним на конференції в Берліні. Також ігри у жанрі roguelite зазвичай поєднують у собі особливості roguelite та інших жанрів, таких як платформери, стратегії в реальному часі, пригодницькі бойовики та інші.

Roguelite ігри зазвичай віддають перевагу більш коротким сесіям, ніж класичні ігри в жанрі roguelike. У той час, коли roguelike ставить на перше місце не досягнення перемоги, а сам процес гри, roguelite ігри зазвичай створюють короткі ігрові заходи, що дають змогу розблокувати нові навички або персонажів та в подальшому з їх допомогою швидко завершити гру.

1.2.Процедурна генерація та її місце в roguelite іграх.

Процедурна генерація[3] – комп'ютерний алгоритм, що автоматично генерує контент, спираючись на попередньо написані інструкції, і таким чином може значно зменшити обсяг роботи, який потрібно виконувати власноруч. Техніка процедурної генерації може бути застосована до безлічі галузей, таких, як анімація, комп'ютерна графіка, 3D-моделювання, створення штучної електронної музики. У розробці комп'ютерних ігор алгоритми процедурної генерації також знайшли своє застосування[4].

Процедурна генерація набула широкого розповсюдження у сфері ігрової розробки. В контексті ігрової розробки "контент", що генерується алгоритмом, може відноситися до широкого спектру речей: графічні текстури, моделі об'єктів, рельєф світу, розташування ігрових об'єктів тощо. Процедурна генерація дозволяє розробникам створювати більше вмісту з меншими зусиллями та ресурсами, а також забезпечує гравцям унікальний ігровий процес. Процедурна генерація почала використовуватися в ігровій розробці ще на початку її розвитку. Тоді, через обмеження пам'яті, ігровий контент часто потрібно було генерувати, бо збереження великого обсягу наперед створених ресурсів було неможливим. До перших ігор з процедурною генерацією контенту можна віднести Rogue (1980), Tetris (1984), Diablo (1996). Структуруючи переваги використання процедурної генерації, можна виокремити наступні пункти:

- Швидке створення контенту, який буде відповідати вимогам дизайнера.
- Можливість створення різноманітного контенту навіть при умові схожих вимог до нього.
- Зменшення часу та грошей, що буде витрачено на розробку гри.

- Потенційна можливість використання процедурної генерації контенту як основи для автоматичної адаптації ігор до гравців.

Процедурна генерація є однією з важливих складових у roguelite іграх. Як було зазначено раніше, roguelite походить від жанру roguelike та зберігає його основні особливості, однією з яких є процедурна генерація оточення, виходячи з того, що дія гри зазвичай відбувається в унікальному середовищі, що повинно гарантувати новий досвід для кожної сесії.

Процедурна генерація оточення в roguelite-іграх є комплексним та складним процесом. Рівні в іграх жанру roguelite (зазвичай підземелля) здебільшого становлять лабіринтоподібне оточення, що складається з викликів, винагород та головоломок, щільно розподілених в ігровому просторі та в часі ігрового сеансу, щоб запропонувати структурований та збалансований ігровий процес. Зважаючи на те, що процедурна генерація оточення це автоматизований процес його дизайну, алгоритми процедурної генерації зустрічаються з тими самими труднощами, що і ручне створення ігрових рівнів. Вони повинні створювати тісний взаємозв’язок між ігровим процесом та ігровим простором, а також підлаштовувати ігровий простір під очікуваний стиль ігрового процесу, запобігаючи небажаним результатам, таким як зникнення частини ігрового контенту через особливості створення оточення, що і стане місцем розташування цього контенту.

Надалі в роботі будуть розглянуті способи генерації оточення (підземелля) в roguelite-іграх, що використовуються для розв’язання проблем дизайну ігрового оточення і його зв’язку з ігровим контентом.

Розділ 2: Способи процедурної генерації оточення та опис обраного алгоритму.

2.1 Види процедурної генерації в roguelite іграх.

Існує багато алгоритмів для генерації оточення (підземелля) в roguelite-іграх. Кожен алгоритм має свій підхід до створення середовища гри. Усі способи процедурної генерації підземель так чи інакше спираються на використання генераторів випадкових чисел, проте ступінь контрольованості випадковості в процесі створення оточення може сильно відрізнятися. Серед розповсюджених алгоритмів процедурної генерації оточення для ігр жанру roguelite можна виокремити:

- Алгоритм Random Walk [5]. Цей алгоритм передбачає випадкове "ходіння" по сітці, в результаті чого формуються шляхи та кімнати. Однією з найпростіших реалізацій цього алгоритму є Drunkard`s Walk[6] (прогулянка п'яниці). Для виконання алгоритму слід обрати клітинку на заповненій сітці та зробити її порожньою. Наступним кроком є вибір випадкового напрямку руху, під час якого всі клітинки на шляху стають порожніми (якщо вони ще не є такими). Ці кроки повторюються до досягнення потрібної кількості порожнього простору. Загалом, порожні клітинки утворюють кімнати (великі групи суміжних порожніх клітин) та коридори (послідовності порожніх клітин з невеликою шириною) між ними. Перевагами цього алгоритму можна назвати простоту реалізації[7], можливість генерувати органічні та звивисті варіанти місцевості.

Недоліком є висока неконтрольованість результатів алгоритму

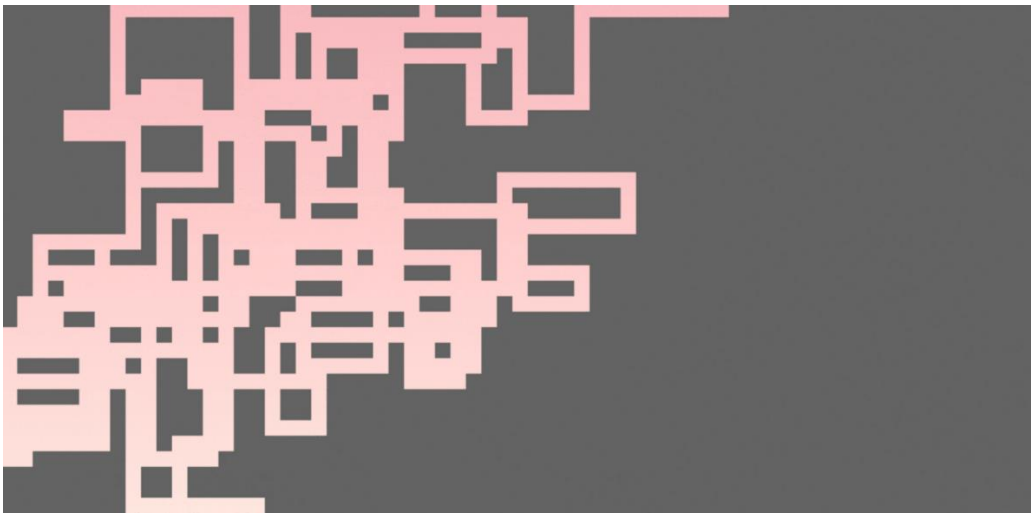

Рис.2 Карта згенерована алгоритмом Drunkard`s walk

- Алгоритм з використанням кліткового автомату (Cellular Automata)[8]. Цей алгоритм можна використовувати для двовимірної або тривимірної сітки клітин. Початково усі клітини мають стан “шлях” або “стіна” , далі задаються правила , які будуть змінювати стани клітин на основі станів їх сусідів. Після цього впродовж певної кількості ітерацій кожна клітина оновлює свій стан на основі правил кліткового автомату. Зазвичай можна використовувати правило , відповідно до якого клітина може стати стіною , якщо навколо неї є певна задана кількість стін. Після певної кількості ітерацій алгоритму згенероване підземелля можна проаналізувати на наявність кімнат та коридорів а також додатково з’єднати нез’єднані частини оточення. Перевагою цього алгоритму є можливість генерувати підземелля з різними структурами відповідно до обраного набору правил, тобто він є досить гнучким. Недоліками є складність контролю над процесом генерації , бо для цього можливо використовувати лише певний набір правил, зміна якого часто може призводити до несподіваних результатів.

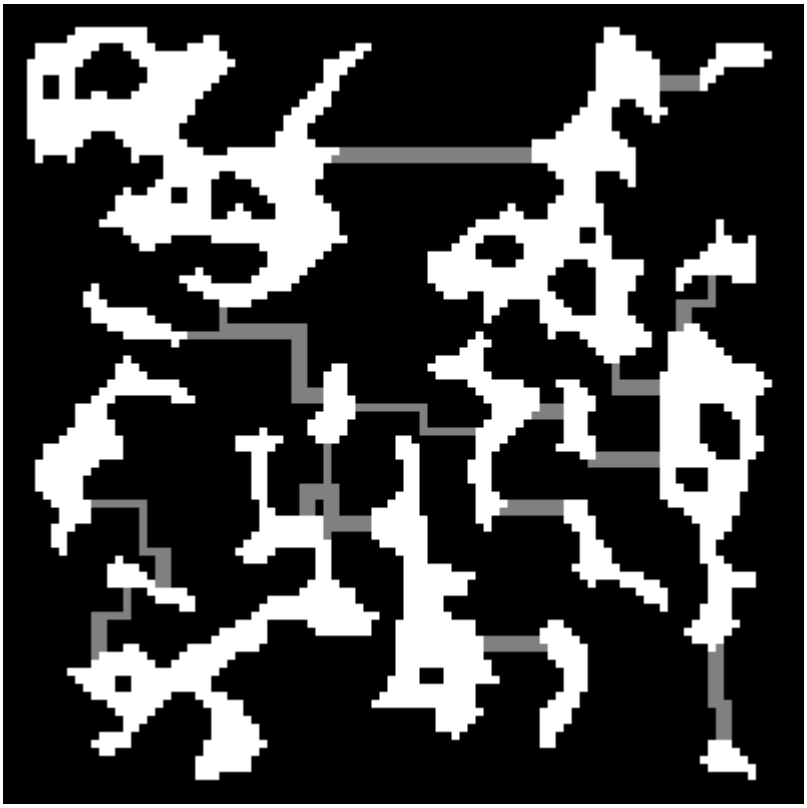

Рис.3 Карта згенерована з використанням кліткового автомата

- Алгоритм BSP (Binary Space Partitioning або двійкове розбиття простору) [9]. Цей алгоритм полягає в рекурсивному діленні простору на менші секції. Після розділення простору на частини, кімнати розташовуються всередині цих секцій і з'єднуються коридорами. Перевагою цього алгоритму є простота реалізації а також можливість створювати збалансоване та структуроване ігрове середовище з різною кількістю великих та малих кімнат. Недоліками є обмеженість кімнат розмірами та формами отриманих в результаті роботи алгоритму секцій.

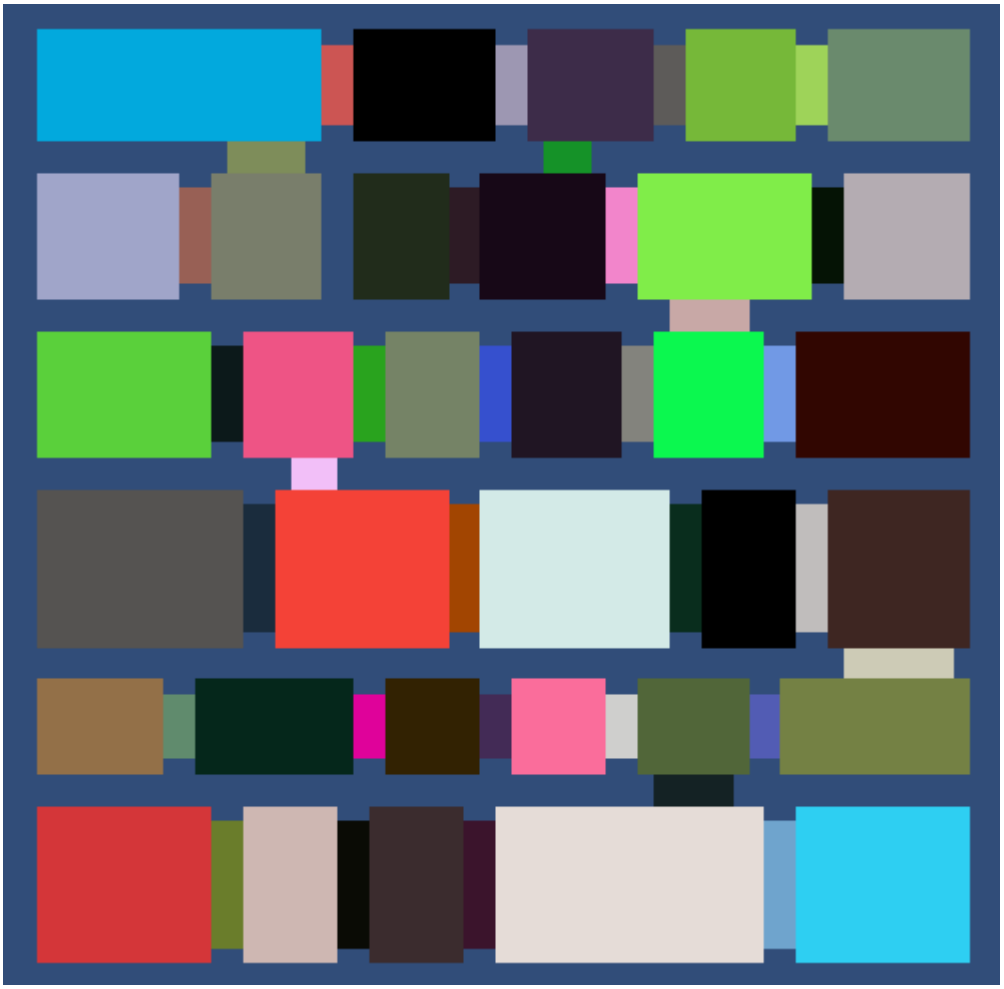

Рис.4 Схема підземелля, згенерованого з алгоритмом BSP.

- Алгоритми генерації на основі сітки (grid-based algorithms)[10] – підхід до створення рівнів, який використовує сітку як базову структуру. Сітка складається з клітин, кожна з яких може являти собою певну частину ігрового середовища. Сітку можна заповнювати за допомогою різних алгоритмів та правил ,що дозволяє створювати різноманітні рівні в великій кількості. Перевагами алгоритму є зручність, налаштованість та контрольованість результату. До недоліків можна віднести змушене прив’язування до сітки та можливість створювати занадто однорідні рівні з дуже схожою структурою , якщо не приділити достатньо уваги налаштуванням алгоритмів заповнення сітки.

Рис.5 Підземелля, згенероване на основі сітки.

- Алгоритми генерації на основі графу (graph-based algorithms)[11] – алгоритми, що використовують структуру графу для створення підземелля. Такі алгоритми мають декілька основних етапів:
    a. Генерація вузлів: Спочатку створюються вузли, які можуть представляти кімнати або інші значущі об'єкти в підземеллі.
    b. З’єднання вузлів за допомогою ребер, які представляють собою коридори/переходи між кімнатами.
    c. Оптимізація графа для забезпечення певних властивостей, таких як зв’язність та відсутність перекриттів кімнат.
    d. Перетворення графа на підземелля шляхом заміни вузлів на кімнати і ребер на коридори.

  Перевагами таких алгоритмів є забезпечення зв’язності між частинами підземелля та контроль над рівнем складності згенерованого ігрового середовища. До недоліків слід віднести обмеженість структури згенерованих локацій в порівнянні з іншими алгоритмами (підземелля можуть отримувати менш органічні та менш випадкові форми).

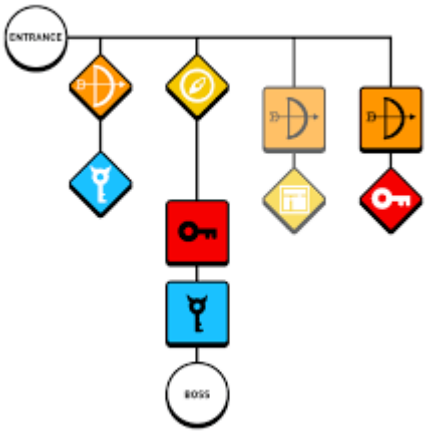

Рис.6 Схематичне зображення графу , що описує підземелля

Вище було розглянуто лише одні з поширених алгоритмів процедурної генерації оточення в roguelite-іграх, проте слід зазначити, що існує ще велика кількість інших підходів до процедурної генерації, які можна використовувати в залежності від технічного завдання та очікуваного результату генерації.

2.2 Обраний алгоритм процедурної генерації оточення.

2.2.1 Загальний огляд та причина вибору

Обраний алгоритм процедурної генерації оточення базується на алгоритмі, використаному в грі Enter The Gungeon[12], випущеній компанією Dodge Roll 2016 року. Цей алгоритм поєднує підходи генерації на основі сітки та на основі графу, що дозволяє створювати цікаві підземелля з контрольованою структурою.

Основною причиною вибору цього алгоритму є його простота в реалізації та здатність генерувати підземелля з контрольованою структурою. Алгоритм передбачає наявність наперед створених об'єктів кімнат (префабів), які мають свій тип, та графів, що контролюють їх

розміщення. Кожен граф має вузли, які відповідають кімнатам певного типу, та ребра, що відповідають коридорам. Завдяки використанню наперед створених графів для визначення розміщення кімнат, алгоритм може гарантувати зв'язність та збалансованість генерованих підземель; наявність наперед створених кімнат дозволяє контролювати їх наповненість та складність. Таким чином генератор повинен розв’язати задачу розміщення кімнат і з’єднання їх між собою, у той час як створення самих кімнат контролюється людиною. Це можна вважати як перевагою, так і недоліком підходу, бо за високий рівень збалансованості та гарну наповненість кімнат доводиться платити збільшенням обсягу роботи, яку повинен виконати дизайнер оточення.

Поєднання підходів з використанням сітки та графів полягає у тому, що алгоритм використовує структуру графа для опису макроструктури підземелля (розташування кімнат та коридорів відносно одне одного) та сітку для мікроструктури (розташування кімнат та коридорів у просторі, розміщення об'єктів, перешкод та деталей кімнат). Це дозволяє легко адаптувати алгоритм під потреби конкретної гри та гарантувати задовільні результати для гравців.

Також алгоритм дозволяє легко масштабувати рівень складності генерованих підземель, шляхом зміни параметрів графа (зміна типів кімнат, їх розташування та кількості), що використовуються в процесі генерації. Це надає більше можливостей для налаштування генерованого середовища та забезпечення бажаного рівня виклику для гравців.

В цілому, алгоритм генерації підземель на основі сітки та графів, використаний у грі "Enter The Gungeon", виявився привабливим для даної роботи через його відносну простоту, зрозумілість, контрольованість та здатність адаптуватися до потреб різних ігрових проектів.

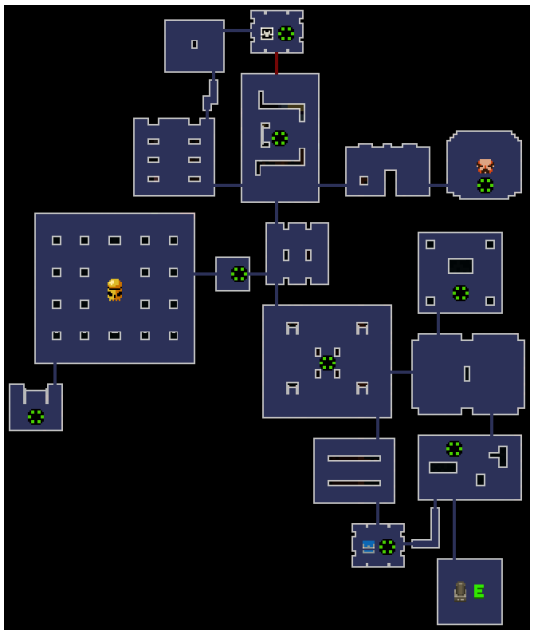

Рис.7 Карта підземелля у грі Enter the Gungeon

2.2.2 Опис алгоритму.

Структуру алгоритму можна описати наступною блок-схемою:

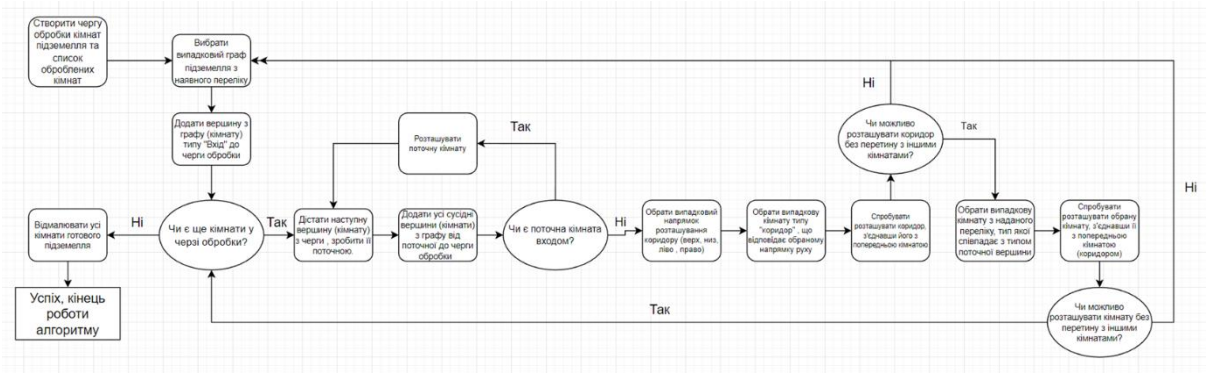

Рис.8 Блок-схема алгоритму.

Перед заглибленням у сам алгоритм потрібно нагадати про деякі наперед створені об’єкти , які будуть ним використовуватися:

- Префаби кімнат (prefabs)[14] – об’єкти Unity, що зберігають інформацію про кімнати на сітці.

- Скриптові об’єкти кімнат (Scriptable objects)[15] – об’єкти Unity, що зберігають іншу інформацію про кімнати, таку як тип та локальні координати.
- Графи рівнів – графи, що контролюють побудову підземелля. Як було зазначено раніше, вершинами графу є кімнати, а ребрами – коридори.

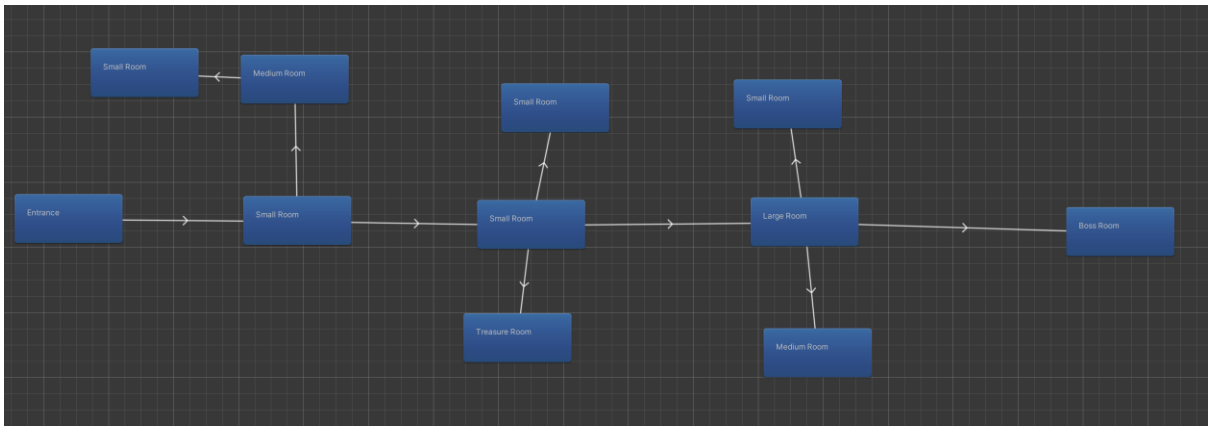

Рис.8 Граф, що описує структуру підземелля.

Роботу генератора можна розділити на 6 основних етапів:

1. Початкові приготування:

   На початку роботи алгоритму створюється черга обробки кімнат підземелля та список оброблених кімнат, готових до відображення. Процес роботи генератора триватиме, доки в черзі на обробку ще будуть наявні необроблені кімнати.

2. Створення входу:

   Надалі з переліку наявних графів випадковим чином обирається один, що стане шаблоном для побудови ігрового оточення. Кожен граф повинен містити кімнату типу “Вхід” (“Entrance”). Тож початково кімната заданого типу буде знайдена в графі та додана до черги обробки.

3. Початок ітеративного процесу, розміщення кімнати-батька:

Далі починається ітеративний процес, що відбуватиметься до моменту, поки побудова підземелля не завершиться успіхом, або не стане неможливою. З черги обробки дістається кімната, яка стає поточною. Відповідно до структури графу, усі сусідні вершини до поточної додаються до черги обробки. Якщо поточна кімната має тип “Вхід”, то вона просто розміщується у просторі (слід зазначити, що “розміщення” в даному контексті ще не передбачає фактичне відображення кімнати на екрані, ми лише обраховуємо і зберігаємо її світові координати відповідно до локальних координат та координат попередньої кімнати, відносно якої буде розміщено поточну, якщо така наявна, і додаємо цю кімнату у список розташованих кімнат) і зберігається як батьківська кімната для наступних кімнат. Після цього алгоритм повернеться до діставання нової поточної кімнати з черги.

4. Додавання коридору:

Якщо ж поточна кімната не є входом, то її потрібно розташувати у просторі, з’єднавши з батьківською кімнатою коридором. Буде обрано випадковий напрямок руху генератора відносно кімнати-батька. Кожна кімната має чотири проходи для з’єднання з коридорами (top / верх, bottom / низ, right /право, left / ліво), напрямок руху визначає з яким з проходів буде з’єднано коридор.

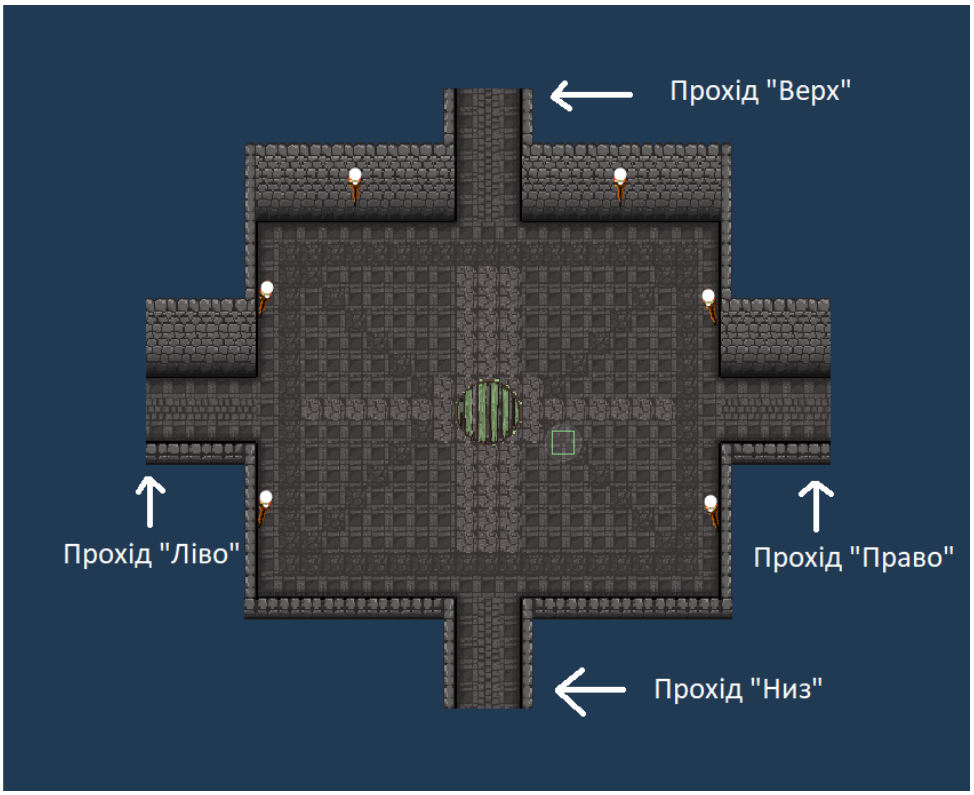

Рис.9 Кімната та типи її проходів.

Коли напрямок руху обрано, із загального переліку кімнат обирається коридор відповідного типу (тип коридору відповідає його напрямку, наприклад left-bottom для коридору, що йде зліва вниз) і відбувається спроба його розташування відносно батьківської кімнати (початок коридору збігається за координатами з відповідним проходом кімнати-батька).

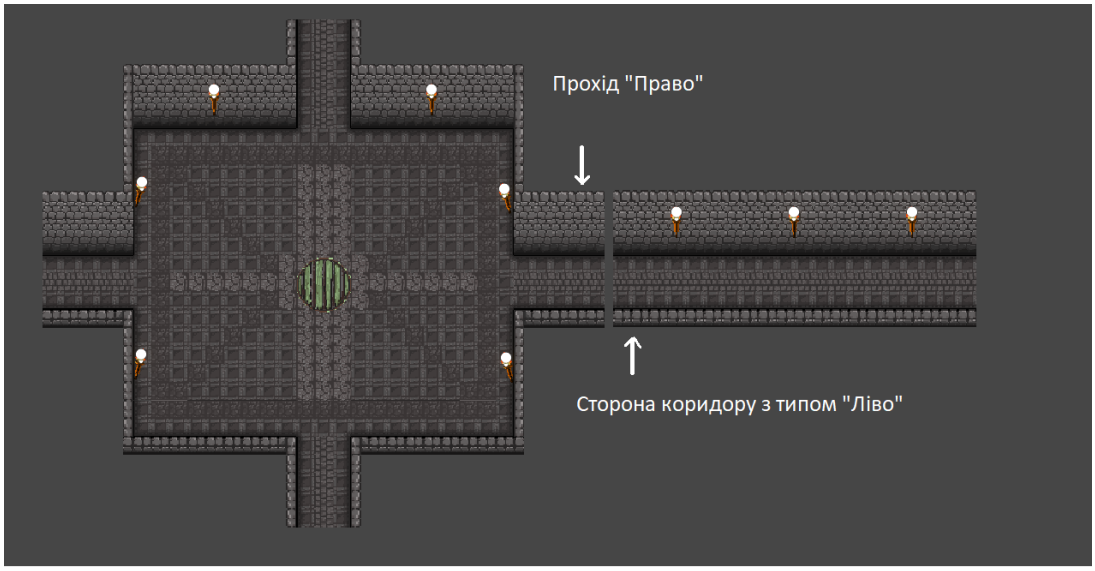

Рис.10 Під'єднання коридора з типом "LeftRight" (ліво-право) до кімнати типу Entrance (вхід)

Якщо коридор неможливо розташувати так, щоб він не перетинався з жодною з попередньо розташованих кімнат, то обирається інший вид коридору, що також можливо з'єднати з проходом кімнати, і спроба розташування повторюється ще декілька разів. У випадку остаточної неможливості розташування коридору, загальне розташування вважається суперечливим і генератор починає побудову підземелля наново з моменту вибору випадкового графу.

5. Розташування поточної кімнати:

Коли коридор було розташовано і з'єднано з батьківською кімнатою, залишається розташувати поточну кімнату і генератор зможе перейти на наступну ітерацію. Генератор обирає випадкову кімнату з наявного переліку скриптових об'єктів (scriptable objects), та намагається розташувати її у просторі, з'єднавши відповідний прохід кімнати з вільною стороною коридору. Як і у випадку з розташуванням коридорів, генератор робить декілька спроб розташування кімнати, перевіряючи чи не перекриває вона одну з раніше оброблених кімнат. У випадку, коли розташування кімнати неможливе, уся поточна структура

підземелля вважається суперечливою і генератор починає роботу наново з моменту обирання випадкового графу. В іншому випадку, поточна ітерація роботи генератора закінчується.

6. Завершення ітерації:

Після кожного циклу обробки поточної кімнати та з'єднання її коридором, генератор перевіряє, чи залишились ще кімнати в черзі обробки. Якщо ще є доступні кімнати, то генератор починає наступну ітерацію своєї роботи, інакше ж побудова підземелля вважається успішною і генератор вимальовує його в ігровому середовищі.

Розділ 3: Розробка.

3.1 Інструменти, використані в розробці.

Для розробки проєкту до курсової роботи я використовував ігровий рушій Unity[13]. Зараз Unity є одним з найпопулярніших та найзручніших засобів створення невеликих ігрових продуктів, бо він надає великий спектр інструментів та можливостей для швидкої та зручної розробки ігор. На даний момент основною та найбільш розповсюдженою мовою програмування, що рекомендовано використовувати разом з Unity є C#, що, на мій погляд, є перевагою рушія, бо C# є дуже потужним механізмом для реалізації складних та гнучких ігрових механік та алгоритмів.

Основними інструментами Unity, що дозволили реалізувати гру з процедурною генерацією підземель, були префаби (prefabs[14]) та скриптові об'єкти (scriptable objects[15]).

Префаби є одним з найважливіших та найрозповсюдженіших інструментів розробки на Unity. Вони дозволяють створювати та зберігати ігрові об'єкти, які піддаються зручному налаштуванню та повторному використанню. Крім того, використання префабів сприяє більшій структурованості проєкту, що полегшує подальший його розвиток. Завдяки ним я зміг створити варіації кімнат та коридорів для ігрового оточення, наповнені різними ігровими компонентами, та зручно використовувати їх кожного разу, коли генератору треба було створити кімнату певного типу або з'єднати кімнати коридорами.

Скриптові об'єкти є зручними контейнерами для даних, що не залежать від конкретного екземпляру класу. Вони були вкрай корисними для зберігання потрібної інформації про об'єкти ігрового світу, такі як розташування проходів у локальних координатах кімнати, локальні

координати місць створення ворогів та інше. Скриптові об’єкти дозволили створити легко масштабовану систему, яка може бути розширена та модифікована у майбутньому. Крім того, використання скриптових об’єктів сприяло чистому та зрозумілому коду, бо дозволило відокремити дані від ігрової логіки. Такий підхід допоміг уникнути великої кількості помилок, а також зробити тестування гри набагато легшим та ефективнішим.

3.2 Огляд розробленої гри.

Розроблена мною гра отримала назву Tomb of the Mage і є представником жанру roguelite. Цільовою платформою для гри є комп’ютери на базі Windows, а аудиторію, на яку націлена гра, складають фанати популярних roguelite-ігор, таких як Enter the Gungeon та The binding of Isaac. Tomb of the Mage переймає багато особливостей, що є традиційними для ігор даного жанру. Такими елементами наприклад можна назвати стандартну схему керування гравцем (переміщення на клавіші WASD , ривок на space, постріли на ліву кнопку миші, пауза на Esc), стандартну схему побудови локацій, де в деяких кімнатах можна знайти підсилювачі, а також де кожен рівень повинен закінчуватися битвою з сильним ворогом. Також особливістю гри можна назвати унікальний візуальний стиль, бо усі графічні компоненти створювались спеціально і виключно для цієї гри.

Основною ідеєю гри є дослідження процедурно згенерованих підземель та подолання усіх могутніх вартових цього підземелля (3 боси гри). Саме завдяки процедурній генерації гра набуває відчуття випадковості та робить кожен прохід гравцем унікальним.

Головне меню гри містить в собі художню ілюстрацію ігрового процесу , також демонструє назву гри та дає можливість розпочати ігровий процес, закінчити гру та змінити налаштування звуку.

Рис.11 Головне меню гри

Під час основного ігрового процесу гравець бачить свого героя, ігрове оточення та стандартні для даного жанру ігор елементи користувацького інтерфейсу (показники здоров'я та магічної енергії, міні-карту, показник обраної зброї).

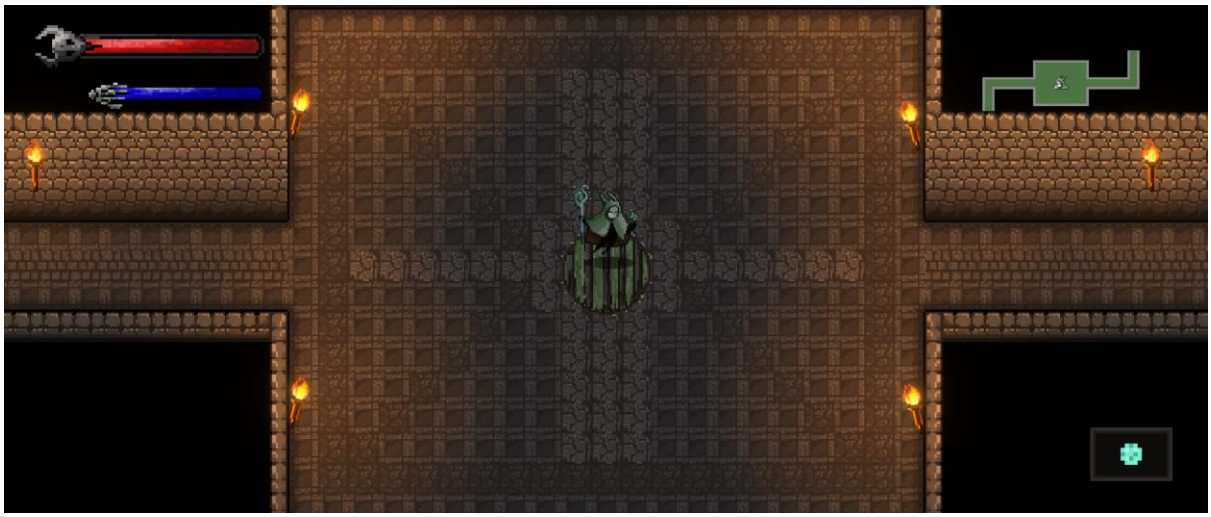

Рис.12 Гравець у початковій кімнаті

Під час розробки Tomb of the Mage значні зусилля було спрямовано на розвиток основних ігрових елементів та механік, щоб, навіть з

урахуванням обмеженості ресурсів під час розробки, створити унікальний ігровий досвід для гравців та основу для подальшого розвитку проекту.

3.3 Огляд класу генератора. Результат.

У процесі розробки гри було створено клас-генератор підземель, що реалізує описаний у пункті 2.2.1 алгоритм процедурної генерації оточення. Даний клас використовується іншими ігровими системами задля створення рівнів гри.

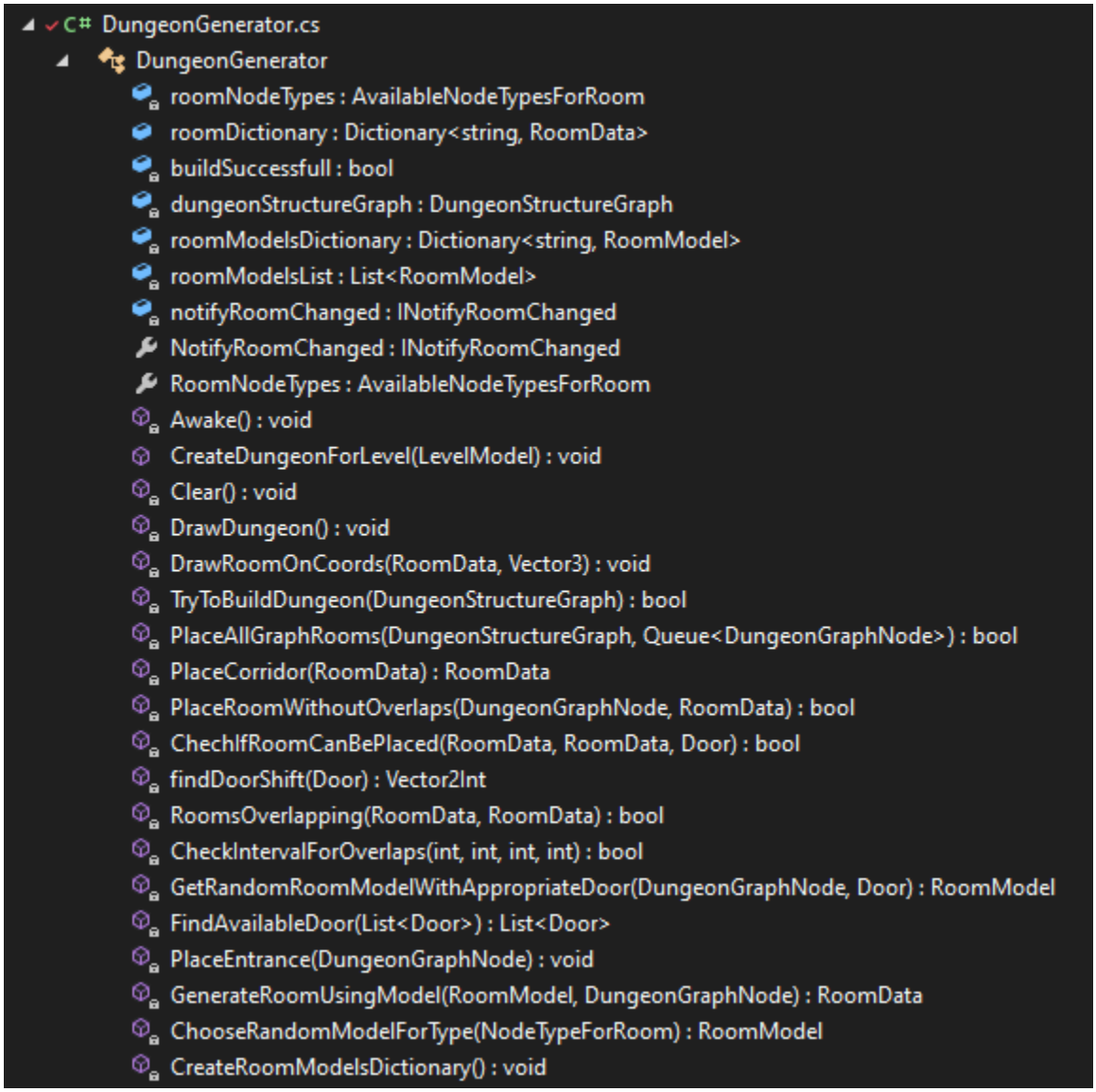

Рис.13 Структура класу-генератора

Серед головних функцій класу слід виокремити:

- CreateDungeonForLevel – головний метод класу, що розпочинає процес побудови підземелля для заданої моделі рівня гри (модель містить в собі список кімнат та граф для побудови рівня). Саме цей метод викликає система ігрових станів (GameStatesSystem) коли потрапляє у стан побудови оточення (DungeonBuildingState).
- TryToBuildDungeon – метод, що виконує одну ітерацію алгоритму побудови підземелля. Він намагається розташувати випадкові кімнати згідно з графом. У випадку успіху повертає true, інакше false.
- PlaceAllGraphRooms – метод, що почергово запускає обробку кожної наступної кімнати (вершини) з графу, щоб додати її до підземелля.
- PlaceEntrance – розташовує першу кімнату підземелля типу Entrance.
- PlaceRoomWithoutOverlaps – метод, що робить декілька спроб розташувати кімнату так, щоб вона не перетиналася зі вже наявними кімнатами підземелля.
- PlaceCorridor – метод, що з'єднує попередню кімнату з коридором. Коридор також розташовується так, щоб не перетинатися з іншими кімнатами та коридорами.
- DrawDungeon – малює підземелля на екрані, розташовуючи відповідні префаби кімнат та коридорів в ігровому світі.

Дані методи описують повний шлях роботи алгоритму. Звичайно ж, з метою запобігання надмірному розростанню окремих методів та забезпечення кращої інкапсуляції окремих функціональних блоків, були

створені допоміжні методи, що сприяло структуризації коду та полегшило його підтримку. Серед таких методів :

- CheckIfRoomCanBePlaced - перевіряє чи можливо розташувати кімнату так, щоб коридором можна було з'єднати її з батьком а також щоб вона не перетиналася з жодною іншою кімнатою чи коридором.
- RoomsOverlapping – перевірка, чи не перетинаються між собою дві кімнати, або кімната та коридор.
- CheckIntervalForOverlaps – порівнює числові інтервали, у яких лежать крайні точки кімнат, щоб дати відповідь на питання чи є між ними перетин.
- GetRandomRoomModelWithAppropriateDoor – обирає випадкову модель кімнати або коридора, що містить прохід заданої орієнтації.
- FindAvailableDoor – знаходить двері (прохід) в кімнаті, що ще не був з'єднаний з коридором.
- GenerateRoomUsingModel – для заданої моделі кімнати знаходить префаб серед наявних.

Результатом стала структурована система, що здатна генерувати підземелля відповідно до структури, вказаної у графі, та з використанням наперед створених префабів та коридорів, доступних для певного ігрового рівня.

Висновок

У ході розробки комп'ютерної гри було продемонстровано різницю між іграми жанрів roguelike та roguelite, досліджено різні алгоритми процедурної генерації підземель у roguelite-іграх, виокремлено та використано зручні механізми Unity для реалізації ігрових проектів.

Під час дослідження стало зрозуміло, що різні підходи до процедурної генерації підземель можуть бути доцільними до використання в залежності від типу результату, якого хоче досягти розробники. В результаті дослідження було обрано алгоритм, що найбільш вдало підійшов для реалізації задуманого типу підземель у грі, а також було продемонстровано, що функціонал Unity та C# дозволяє створювати сучасні комп'ютерні ігри зі складними механіками та алгоритмами. Було створено гру, що має свій власний візуальний стиль, цікаві механіки та неповторні процедурно згенеровані рівні.

Використані джерела

[1].G2A.COM.(2022). Roguelike vs Roguelite | Do you know the
URL: https://www.g2a.com/news/features/roguelike-vs-roguelite-do-you-know-the-difference/

[2]. Stegen B. What are roguelike and roguelite video games?
URL: https://www.makeuseof.com/what-are-roguelike-and-roguelite-video-games/

[3] Computer Hope. (2022). Procedural Generation.
URL: https://www.computerhope.com/jargon/p/procedural-generation.htm

[4] Togelius, Emil Kastbjerg, David Schedl and Georgios N. Yannakakis. (2011). What is Procedural Content Generation?Mario on the borderlineJulian
URL:
https://www.researchgate.net/publication/228620622_What_is_Procedural_Content_Generation_Mario_on_the_borderline

[5]Subham Datta. (2023). What Is a Random Walk?
URL: https://www.baeldung.com/cs/random-walk

[6] Procedural Content Generation Wiki. (2011). Drunkard Walk.
URL: http://pcg.wikidot.com/pcg-algorithm:drunkard-walk

[7] Margaret Gaston. (2022). How to code your own procedural dungeon map generator using the Random Walk Algorithm.
URL: https://copyprogramming.com/howto/how-to-code-your-own-procedural-dungeon-map-generator-using-the-random-walk-algorithm

[8] Roland van der Linden, Ricardo Lopes and Rafael Bidarra. (2014). Procedural generation of dungeons , 4-6.
URL: https://graphics.tudelft.nl/Publications-new/2014/LLB14/Procedural.pdf

[9] Gonzalo Uribe. (2019). Dungeon Generation using Binary Space Trees
URL: https://medium.com/@guribemontero/dungeon-generation-using-binary-space-trees-47d4a668e2d0

[10] RogueBasin. (2011). Grid Based Dungeon Generator.
URL: https://medium.com/@guribemontero/dungeon-generation-using-binary-space-trees-47d4a668e2d0

[11] Thomas Smith, Julian Padget, Andrew Vidler. (2018). Graph-based Generation of Action-Adventure Dungeon Levels using Answer Set Programming.
URL: https://purehost.bath.ac.uk/ws/portalfiles/portal/185972557/TSmithPureGBGwASP.pdf

[12] BorisTheBrave.Com. (2019). Dungeon Generation in Enter The Gungeon.

URL: https://www.boristhebrave.com/2019/07/28/dungeon-generation-in-enter-the-gungeon/

[13] Unity.com. (n.d). Scripting in Unity for experienced programmers.

URL: https://unity.com/how-to/programming-unity

[14] Unity.com. (n.d). Prefabs.

URL: https://docs.unity3d.com/Manual/Prefabs.html

[15] Unity.com. (n.d). ScriptableObject.

URL: https://docs.unity3d.com/Manual/class-ScriptableObject.html