

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Гра "Пекельний калібр" у жанрі Roguelike»

Виконав студент групи ПЗ-21-5
ЯНУШЕВИЧ Дмитро Петрович

Керівник роботи:
ВЛАСЕНКО Олег Васильович

Рецензент:
САВІЦЬКИЙ Роман Святославович

Житомир – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна ВАКАЛЮК

«26» лютого 2025р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: ЯНУШЕВИЧ Дмитро Петрович

Керівник роботи: ВЛАСЕНКО Олег Васильович

Тема роботи: «Гра "Пекельний калібр" у жанрі Roguelike»,

затверджена Наказом закладу вищої освіти від «14» лютого 2025р., №61/с

Вихідні дані для роботи:

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів,
що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Власенко О.В.	27.02.2025	27.02.2025
2	Власенко О.В.	27.02.2025	27.02.2025
3	Власенко О.В.	27.02.2025	27.02.2025

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з гри "Пекельний калібр" у жанрі Roguelike та пояснювальної записки. Пояснювальна записка до випускної роботи загально містить 75 сторінки, з них 56 друкованого тексту та 19 додатків, в документі 31 ілюстрація, 10 таблиць та 31 найменування використаних джерел.

Метою роботи є розробка функціональної та захоплюючої 2D-гри в жанрі roguelike з використанням рушія Unity та мови програмування C#, яка реалізує процедурну генерацію рівнів, бойову систему, систему прокачування персонажа, а також забезпечує зручний та інтуїтивний інтерфейс для гравця.

Архітектура гри побудована з урахуванням принципів модульності та розділення відповідальностей, що забезпечує зручну підтримку та масштабованість проекту. Основна увага приділена процедурній генерації рівнів, системі бою, прокачуванню персонажа та інтерактивній взаємодії з ігровим середовищем.

Інтерфейс гри є інтуїтивно зрозумілим і адаптованим до потреб гравця, що дозволяє легко орієнтуватися у світі гри, взаємодіяти з ворогами, предметами та виконувати дії без ускладнень.

КЛЮЧОВІ СЛОВА: ROGUELIKE, UNITY, C#, ПРОЦЕДУРНА ГЕНЕРАЦІЯ, 2D, ІГРОВИЙ ДИЗАЙН, БОЙОВА СИСТЕМА, ІНТЕРФЕЙС.

					ІПЗ.КР.Б-121.25 - ПЗ								
Змн.	Арк.	№ докум.	Підпис	Дата	Гра "Пекельний калібр" у жанрі Roguelike				Лім.	Арк.	Аркушів		
Розроб.		Янушевич Д. П.										4	75
Керівник		Власенко О.В.											
Н. контр.													
Затвердив					ФІКТ, гр. ІПЗ-21-5								

Зміст

РЕФЕРАТ	3
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	5
ВСТУП	6
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ В РОЗРОБЦІ ГРИ В ЖАНРІ ROGUELIKE.....	8
1.1. Постановка задачі	8
1.2. Аналіз аналогів програмного продукту	11
1.3. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	20
Висновки до першого розділу.	22
РОЗДІЛ 2. ПРОЕКТУВАННЯ ГРИ "ПЕКЕЛЬНИЙ КАЛІБР" У ЖАНРІ ROGUELIKE	23
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи.....	23
2.2. Розробка системи збереження даних	36
2.3. Проектування та реалізація алгоритмів роботи системи.....	37
2.4 Реалізація функціоналу гри «Пекельний кал» в жанрі Roguelike.....	39
Висновки до другого розділу.....	42
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ ДОДАТКУ	43
3.1 Порядок встановлення та налаштування системи.....	43
3.2 Структура інтерфейсу та порядок взаємодії з грою	43
3.3. Тестування програмного продукту	51
Висновки до третього розділу	52
ВИСНОВКИ.....	53
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	54
Додатки.....	57

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Рогалик – Roguelike, це жанр відеоігор, що характеризується процедурно згенерованими рівнями, високою складністю та перманентною смертю персонажа. Назва походить від гри Rogue (1980), яка стала основою для жанру. Гравці зазвичай досліджують підземелля, збирають здобич і борються з ворогами, починаючи з початку після кожної поразки.

ASCII-графіка – форма образотворчого мистецтва, що використовує символи ASCII на моноширинному екрані комп'ютерного терміналу (термінальний сервер) або принтера для представлення зображень.

Sci-fi – наукова фантастика.

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		5

ВСТУП

Актуальність теми. У сьогоденному світі відеоігри вже давно стали значущою складовою цифрового життя людини та однією з найбільш стрімко розвиваючихся сфер розваг у світі загалом. Зокрема, жанр Roguelike за останнє десятиліття зазнав значного відродження завдяки своїй високій реіграбельності, складності та стратегічній глибині. За останній декілька років цей жанр поповнився безліччю нових представників. Популярність таких ігор пояснюється не лише бажанням гравців випробувати свої навички в умовах нестандартних викликів з випадковими умовами, але й прагненням до унікального досвіду в кожному проходженні, завдяки механіці процедурної генерації рівнів.

2D гра «Пекельний калібр» поєднує ключові елементи жанру Roguelike — процедурну генерацію рівнів, постійні випробування і продумування своїх кроків наперед, систему перманентної смерті персонажа — з оригінальною тематикою та нестандартними бойовими механіками. Такий підхід дозволяє створити захопливий геймплей, здатний зацікавити широку аудиторію — від хардкорних гравців до шанувальників інді-ігор.

Метою випускної кваліфікаційної роботи бакалавра є розробка гри в жанрі Roguelike. Для досягнення цієї мети необхідно провести аналіз жанру Roguelike, визначити його характерні особливості, ігрові механіки та розглянути популярних представників. Після цього слід здійснити вибір засобів проектування, які найкраще підходять для реалізації гри. Далі потрібно розробити архітектуру гри, яка включатиме механіку процедурної генерації рівнів, бойову систему та інтерфейс користувача. Завершальним етапом є тестування гри з метою виявлення та усунення помилок, а також оцінювання якості ігрового процесу.

Для досягнення цієї мети було поставлено наступні завдання:

1. Проаналізувати жанр Roguelike, його особливості, ігрові механіки та популярних представників.

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

2. Вибрати інструменти та середовище розробки, які найкраще відповідають вимогам проєкту.
3. Спроекувати архітектуру гри, включаючи систему процедурної генерації, бойову систему та користувацький інтерфейс.
4. Реалізувати гру відповідно до вимог, із забезпеченням високої реіграбельності та зручності взаємодії.
5. Провести тестування гри з метою виявлення помилок і вдосконалення ігрового процесу.

Об'єктом дослідження є процес розробки відеоігор у жанрі Roguelike, зокрема створення ігрових світів за допомогою процедурної генерації рівнів, впровадження сучасних бойових механік, системи збереження прогресу гравця, а також розробка візуально привабливого та зручного інтерфейсу користувача. Також об'єкт охоплює загальні принципи побудови геймдизайну в умовах високої реіграбельності та динамічного ігрового процесу.

Предметом дослідження є методи та інструменти реалізації Roguelike-ігри «Пекельний калібр», з особливим акцентом на механізми процедурної генерації ігрових рівнів. У дослідженні розглядаються також алгоритми створення бойової системи, способи інтеграції збереження прогресу, побудова логіки взаємодії гравця з ігровим середовищем, підходи до UI/UX-дизайну, вибір архітектури проєкту та засобів розробки, а також тестування для забезпечення якісного ігрового досвіду.

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ В РОЗРОБЦІ ГРИ В ЖАНРІ ROGUELIKE

1.1. Постановка задачі

Метою кваліфікаційної роботи бакалавра є розробка гри «Пекельний калібр» у жанрі Roguelike із використанням сучасних підходів та технологій розробки, що забезпечить можливість швидких регулярних оновлень з додаванням нового контенту, масштабування та подальшого вдосконалення проєкту з боку розробника. З боку користувача гра має забезпечити зручний, захопливий та комфортний ігровий досвід.

Roguelike, тобто rogue-подібні ігри — це піджанр рольових відеоігор, визначними особливостями якого є випадкове, процедурне створення рівнів, перманентна смерть персонажа у випадку поразки. Менш характерними є покроковий ігровий процес, плиткова (тайлова) або ASCII-графіка.

Цей жанр бере свій початок від гри Rogue, випущеної у 1980 році, яка й дала назву всьому напрямку. Упродовж десятиліть Roguelike-ігри еволюціонували, зберігаючи основні принципи, проте активно адаптуючись до нових технічних можливостей та гравецьких очікувань.

Сучасна класифікація Roguelike-ігор нерідко ґрунтується на так званих «Берлінських критеріях», сформульованих на міжнародній конференції Roguelike-розробників у 2008 році, саме ця конференція ледь не вперше дала хочаб скількись точне визначення жанру Roguelike, адже все своє існування воно було туманним. Вони охоплюють такі ключові характеристики, як процедурна генерація, перманентна смерть, тактичний геймплей, складна взаємодія з об'єктами та орієнтація на ресурс-менеджмент. Хоча багато сучасних ігор не повністю дотримуються цих критеріїв, вони зберігають основний дух жанру, доповнюючи його новими механіками, візуальними стилями та системами прогресу, що дозволяє ширше охопити аудиторію гравців.

Roguelike-жанр сьогодні активно використовується не лише в інді-сегменті, але й у великих проєктах з залученням великої кількості виробничих

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

ресурсів, завдяки своїй здатності створювати високий рівень реіграбельності, унікальний досвід у кожному проходженні та мотивацію для самовдосконалення гравця, що дає свої плоди і робить реально якісні ігри цього жанру популярними. Саме ці особливості стали основою концепції гри «Пекельний калібр», яка має за мету поєднати класичні принципи Roguelike з індивідуальним підходом до бойової системи, дизайну рівнів та взаємодії з користувачем.

Більш розгорнено розглянемо критерії жанру – Roguelike має низку ключових критеріїв, які формують його унікальність та відрізняють від інших типів ігор. Однією з найхарактерніших особливостей є процедурна генерація. У таких іграх рівні, картки, місії та інші елементи створюються випадковим чином, що забезпечує унікальність кожного проходження. Завдяки цьому гравець щоразу стикається з новими викликами, що підвищує реіграбельність і не дозволяє грі стати передбачуваною.

Ще одним важливим аспектом є перманентна смерть. У Roguelike-іграх смерть персонажа означає кінець поточного проходження, і гравець не має можливості відновити прогрес із попереднього моменту. Це створює додаткову напругу і змушує гравця бути обережним, відповідально ставитися до кожного рішення, адже ціна помилки — втрата всього досягнутого прогресу.

Тактичний геймплей також є невід’ємною частиною ігор цього жанру. Успішне проходження потребує стратегічного планування та продумування кожного кроку, оскільки навіть незначне рішення може мати серйозні наслідки. Це робить геймплей глибоким і вимагає високого рівня концентрації та аналітичного мислення.

Крім того, Roguelike-ігри зазвичай пропонують складну систему взаємодії з об’єктами та середовищем. Гравець може використовувати предмети, пастки, елементи ландшафту, взаємодіяти з NPC, що відкриває додаткові тактичні можливості й робить геймплей багатогранним.

Ще одним визначальним елементом є орієнтація на ресурс-менеджмент. Гравець повинен ефективно керувати обмеженими ресурсами, такими як

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

здоров'я, їжа, зброя, боєприпаси або інші знаряддя. Раціональне використання цих ресурсів є критично важливим для виживання в умовах високої складності гри. Усе це разом формує характерну для жанру атмосферу напруги, стратегічності й постійного виклику.

Повертаючись до програмного продукту що розробляється, в результаті треба створити продукт який відповідає всім вище переліченим критеріям. Також важливою складовою розробки є продумування привабливого дизайну, що буде сприяти більшому зануренню в геймплей. За планом створення дизайну в стилі Sci-fi.

Основними етапами розробки 2D гри в жанрі Roguelike є кілька взаємопов'язаних процесів. Перш за все виконується розробка дизайну гри, що включає створення загальної концепції, визначення атмосфери, стилістики та візуального стилю, а також проєктування інтерфейсних елементів, таких як меню та панелі статистики. Одночасно формується дизайн рівнів, карт і об'єктів, які згодом будуть створюватися за допомогою процедурної генерації.

Наступним кроком є побудова архітектури клієнтської частини гри. На цьому етапі реалізується основна ігрова механіка, зокрема керування персонажем і бойова система. Впроваджується процедурна генерація рівнів та інших динамічних елементів гри, а також проводиться оптимізація процесів рендерингу для забезпечення стабільної роботи гри на різних пристроях.

Далі йде розробка основного функціоналу. Створюється логіка ігрових механік, включаючи переміщення персонажа, взаємодію з об'єктами та NPC, а також сам процес бою. Імплементується повноцінна система процедурної генерації, яка охоплює об'єкти, ворогів і ресурси. Також реалізовується бойова система з тактичними елементами, система інвентарю, механізми перманентної смерті, а також система покарань і винагород, які формують мотивацію гравця.

Завершальним етапом є тестування гри. Воно включає функціональне тестування, спрямоване на перевірку правильності реалізації ігрових механік, інтерфейсу та процедурної генерації. Проводиться тестування графічного

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

інтерфейсу, що дозволяє оцінити зручність і зрозумілість взаємодії з користувачем. Крім того, здійснюється виявлення та виправлення багів, а також загальна оптимізація продуктивності гри.

1.2. Аналіз аналогів програмного продукту

Розробка гри подібного наповнення та жанру потребує глибокого аналізу та дослідження конкурентів на ринку. Оскільки ігри в жанрі Roguelike досить стрімко розвиваються і в цьому жанрі з'являється все більше і більше ігор, проведемо детальний аналіз наявних популярних ігор жанру Roguelike.

Почнемо з гри The Binding of Isaac.



Рис. 1.1. The Binding of Isaac

Це культова 2D-гра в жанрі roguelike, розроблена Едмундом МакМілленом і Флораном Хімслем. Вона була випущена у 2011 році для Microsoft Windows, а пізніше портована на інші платформи, включаючи PlayStation 4, Xbox One, Nintendo Switch та iOS.

Гравець керує Ісааком або одним із шести доступних персонажів, які намагаються втекти від матері, що отримала божественне повеління принести його в жертву. Події гри розгортаються у підвалі будинку, де кожен рівень — це процедурно згенерована кімната, наповнена ворогами, пастками та таємницями.

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

Ісаак стріляє сльозами, збирає предмети, що змінюють його вигляд і здібності, та бореться з босами.

Гра The Binding of Isaac має низку переваг, які зробили її однією з найпопулярніших у жанрі Roguelike. Однією з головних переваг є висока реіграбельність: завдяки процедурній генерації рівнів, предметів і ворогів, кожне нове проходження суттєво відрізняється від попереднього, що постійно стимулює гравця повертатися до гри. Кожен запуск гри відкриває нові виклики та ситуації, які важко передбачити наперед.

Ще однією сильною стороною є глибока система взаємодії предметів. У грі представлено велику кількість артефактів, які мають унікальні ефекти, а їхні комбінації можуть створювати несподівані й іноді дуже потужні результати, що суттєво змінюють стиль проходження. Це дозволяє експериментувати з різними підходами до бою та розвитку персонажа.

Крім того, The Binding of Isaac вирізняється масштабним контентом. У ній присутня величезна кількість ігрових персонажів, предметів, ворогів, босів і секретів, що забезпечує тривалий і захоплюючий ігровий процес навіть після десятків годин гри. Геймплей є складним, але водночас справедливим: кожна поразка — це результат дій самого гравця, а не несправедливих обставин, що мотивує вдосконалювати навички та шукати ефективні стратегії.

Важливою перевагою є також підтримка модифікацій. Активна спільнота гравців створює моди, які додають нових персонажів, предмети, рівні та інші елементи, тим самим розширюючи можливості гри та дозволяючи кожному гравцеві персоналізувати досвід відповідно до власних уподобань.

Втім, гра має і певні недоліки. Одним із них є відсутність детального опису предметів під час першого знаходження. Це може викликати труднощі, особливо для новачків, які не знають, як саме вплине новий предмет на персонажа. Крім того, деякі гравці з часом відзначають монотонність геймплею — попри значну різноманітність елементів, загальна структура проходжень може почати здаватися повторюваною після тривалого часу.

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		12

Наступним розглянемо Hades.

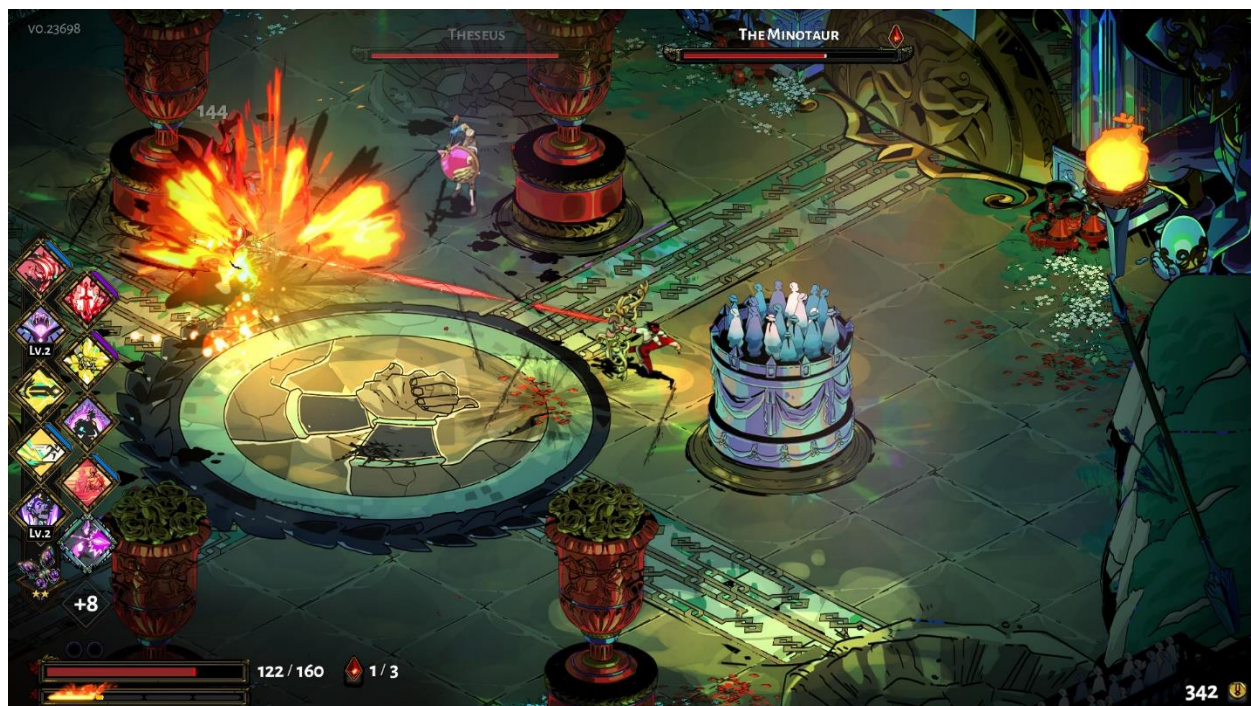


Рис. 1.2. Hades

Це динамічна roguelike-гра в жанрі екшн-RPG, розроблена Supergiant Games та випущена у вересні 2020 року. Гравець керує Загреєм, сином Аїда, який намагається втекти з підземного світу, щоб знайти свою матір, Персефону. Гра поєднує швидкий бій, глибоке оповідання та елементи процедурної генерації, що забезпечує унікальний досвід у кожному проходженні.

Гра Hades має низку вагомих переваг. Однією з найсильніших сторін є захоплюючий бойовий процес. Гра пропонує динамічні, швидкі бої з великою кількістю зброї, кожна з яких має унікальний стиль ведення бою. Крім того, система божественних благословень (boons) дозволяє гравцю налаштовувати стиль гри під власні вподобання, експериментуючи з різними комбінаціями здібностей.

Ще однією важливою перевагою є глибоке оповідання та поступовий розвиток персонажів. Кожна нова спроба втечі з підземного світу відкриває нові сюжетні лінії та діалоги, що розкривають взаємини між персонажами. Таким чином, навіть програші є частиною великої, захопливої історії, яка мотивує повертатися до гри знову і знову.

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

Висока реіграбельність також є характерною рисою Hades. Завдяки процедурній генерації рівнів і різноманітності божественних благ, кожне проходження має унікальний характер. Гравець отримує щоразу новий досвід, що стимулює до багаторазового проходження гри.

Візуальний стиль гри заслуговує окремої уваги. Гра має виразну, художньо стилізовану графіку та насичену палітру кольорів, яка чудово поєднується з атмосферним саундтреком. Це створює сильне відчуття занурення у світ грецької міфології, представлений у сучасній інтерпретації.

Водночас, Hades має і певні недоліки. З часом деякі гравці можуть відчути монотонність, адже хоч рівні й створюються процедурно, структура проходжень залишається схожою, з повторюваними кімнатами та незмінною послідовністю босів. Це може знижувати відчуття новизни під час довготривалої гри.

Ще одним обмеженням є слабка взаємодія з оточенням. Незважаючи на глибокий сюжет і розвиток персонажів, можливості взаємодії з навколишнім світом є доволі обмеженими. Середовище в основному виконує декоративну та функціональну роль у рамках бою, а не як повноцінна частина геймплейної взаємодії.

Далі на огляді Enter The Gungeon – це динамічна 2D-гра в жанрі bullet hell roguelike, розроблена студією Dodge Roll та випущена Devolver Digital 5 квітня 2016 року. Гравець керує одним із кількох персонажів, відомих як «Gungeoners», які досліджують процедурно згенеровані підземелля, борються з ворогами та шукають легендарну зброю, здатну «вбити минуле».

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		14

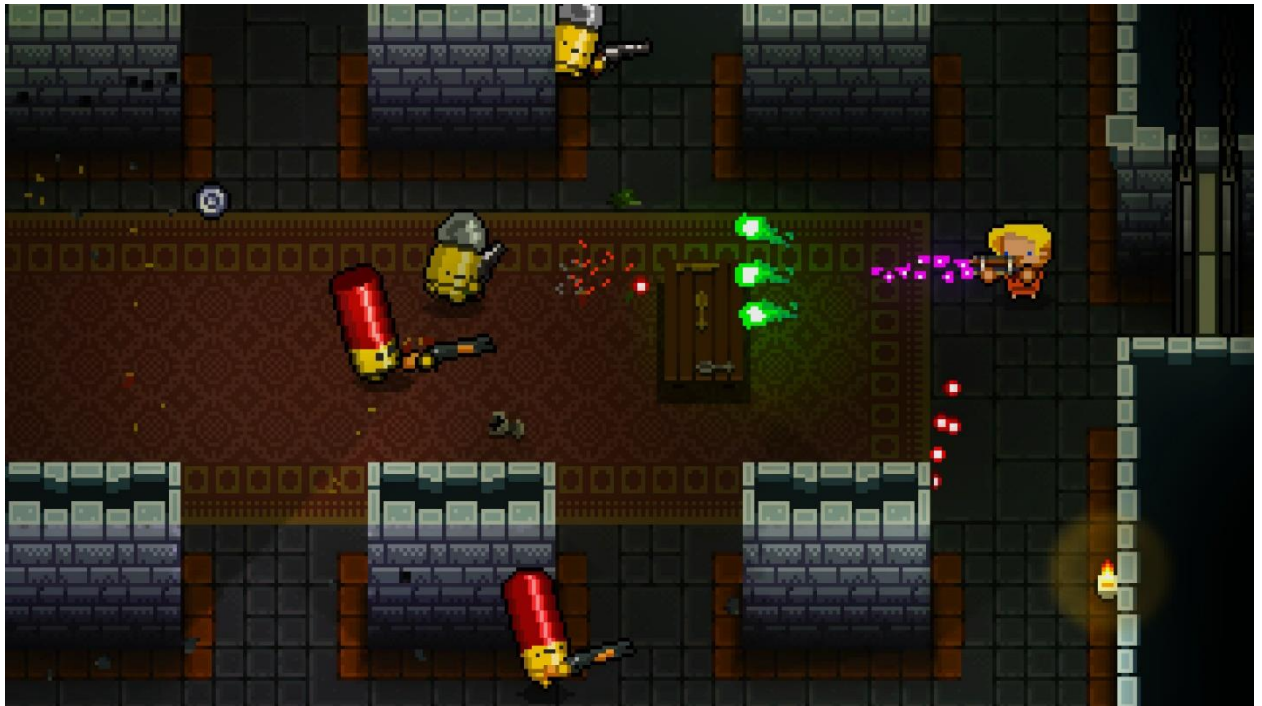


Рис. 1.3. Enter The Gungeon

Enter the Gungeon вирізняється швидким та захоплюючим бойовим процесом. Битви у грі інтенсивні, з великою кількістю ворогів і куль, що літають у різні боки. Успіх залежить від точності, реакції та правильного використання механіки перекатів, яка дозволяє ухилятися від ворожих атак. Великий вибір зброї й предметів — ще одна сильна сторона гри. У ній представлено сотні унікальних, креативних і часто гумористичних видів зброї, що додає геймплею глибини й різноманіття.

Процедурна генерація рівнів забезпечує кожному проходженню унікальність. Кожна нова спроба — це нові кімнати, вороги, боси та предмети, що підтримує інтерес і сприяє високій реіграбельності. Додатковим плюсом є наявність локального кооперативного режиму, що дозволяє грати удвох на одному екрані, спільно проходячи підземелля, що робить гру ще цікавішою у компанії.

Проте, Enter the Gungeon має і свої недоліки. Найбільш очевидним з них є висока складність. Гра вимагає від гравця швидкої реакції, точності та вміння адаптуватися до складних ситуацій. Це може бути занадто складним для новачків

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

або тих, хто не звик до жанру bullet hell, де екран постійно заповнений кулями й вибухами.

Далі розглянемо Dead Cells.



Рис. 1.4. Dead Cells

Це динамічна 2D-гра в жанрі roguelike, натхненна Metroidvania, розроблена французькою студією Motion Twin. Випущена в 2018 році, вона поєднує швидкий платформерний екшн з процедурно згенерованими рівнями та елементами "souls-lite" бою. Гравець керує істотою, яка володіє тілами померлих ув'язнених, намагаючись вибратися з величезної в'язниці, що постійно змінюється.

Гра Dead Cells має кілька важливих переваг. По-перше, її бойовий процес є інтуїтивно зрозумілим і швидким. Бої відзначаються плавною анімацією та чітким управлінням, що дозволяє гравцям оперативно реагувати на атаки ворогів та ефективно ухилятися від них. По-друге, гра вирізняється якісним візуальним стилем — її графіка стилізована і дуже приваблива, що створює особливу атмосферу.

Разом із тим, Dead Cells має і недоліки. Одним із них є те, що повторне проходження перших локацій може викликати втому у гравців. Через

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		
						16

необхідність починати з самого початку навіть після багатьох годин гри, багато хто може втомитися від повторення одних і тих самих перших кількох рівнів, особливо якщо мета — дістатися до пізніших босів чи секретів.

Rogue Legacy – це 2D-платформер у жанрі roguelite, розроблений студією Cellar Door Games і випущений у 2013 році. Гра поєднує елементи класичних платформерів із процедурною генерацією рівнів та унікальною механікою спадковості. Після смерті персонажа гравець обирає одного з трьох нащадків, кожен з яких має випадкові класи та особливості, що впливають на геймплей. Зібране золото використовується для покращення замку та розблокування нових можливостей для наступних поколінь героїв.



Рис. 1.5. Rogue Legacy

Гра Rogue Legacy має одну з ключових переваг — унікальну систему спадковості. Кожен новий персонаж отримує випадкові риси, такі як дальтонізм чи гігантизм, що не лише додає варіативності в ігровий процес, але й приносить гумор і несподіванки під час проходження.

Серед недоліків Rogue Legacy варто зазначити повторюваність ворогів. Після певної кількості забігів вороги стають одноманітними, адже їхні посилення часто обмежуються лише зміною кольору та показників шкоди. Також

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		17

спостерігається нерівномірний баланс класів — деякі персонажі значно ефективніші за інших, що впливає на вибір гравця та може зменшувати різноманітність стратегій під час проходження гри.

Rogue — це легендарна текстова (ASCII) комп'ютерна гра, що побачила світ ще у далекому 1980 році. Вона була створена трьома ентузіастами: Майклом Тойнманом, Гленном Віхманом та Кеном Арнольдом.

Попри свою зовнішню простоту, Rogue стала справжнім проривом у світі ігрової індустрії та фактично започаткувала цілий жанр, який сьогодні відомий як roguelike.

Саме ця гра заклала фундаментальні принципи, які згодом стануть канонічними для багатьох майбутніх хітів: процедурна генерація рівнів, перманентна смерть персонажа, покроковий геймплей та постійна непередбачуваність. Rogue по праву вважається прабатьком усіх ігор цього жанру, залишаючи глибокий слід в історії геймдизайну.

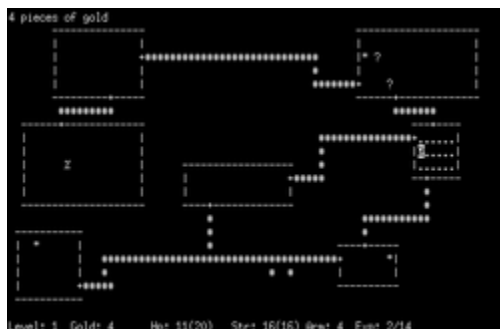


Рис. 1.6. Rogue

Гра Rogue має кілька важливих переваг. По-перше, вона використовує процедурну генерацію, завдяки чому кожне нове проходження є унікальним — рівні створюються випадковим чином, що забезпечує різноманітність ігрового процесу. По-друге, ігрова механіка досить проста: ASCII-графіка і покроковий геймплей позбавлені зайвих елементів, що дозволяє зосередитися виключно на суті гри.

Разом з тим, Rogue має і свої недоліки. Найпомітнішим є відсутність графіки — гра використовує лише текст і символи, що для сучасного гравця може виглядати як старе DOS-вікно з хаотичним набором знаків. Крім того, у грі

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		18

майже нічого не пояснюється, тож без зовнішніх гайдів складно розібратися, що потрібно робити і яке значення мають різні предмети.

Проаналізувавши ринок в цьому жанрі можна прослідкувати декілька ключових, і що важливо, спільних як мінусів так і плюсів.

Таблиця 1.1

Порівняльна таблиця аналогів

Гра	Основний сюжет / тема	Переваги	Недоліки	Особливості / Додаткові відомості
The Binding of Isaac	Ісаак тікає від матері, що хоче його вбити	Висока реіграбельність, глибока система предметів, великий контент, підтримка модів	Відсутність описів предметів для новачків, з часом монотонність	Величезна кількість контенту, складний, але справедливий геймплей
Hades	Загрей тікає з підземного світу, шукає матір	Захоплюючий бій, глибокий сюжет, висока реіграбельність, стильна графіка	Монотонність рівнів, обмежена взаємодія з оточенням	Сильне оповідання, атмосферний саундтрек
Enter The Gungeon	Gungeoners шукають зброю, що вбиває минуле	Інтенсивні бої, великий вибір зброї, локальний кооператив	Висока складність, може бути складно для новачків	Кооперативний режим на одному екрані
Dead Cells	Істота намагається вибратись із в'язниці	Швидкий і плавний бойовий процес, приваблива графіка	Повторення перших рівнів викликає втому.	Поєднання roguelike та metroidvania
Rogue Legacy	Спадкоємці проходять замок	Система спадковості з випадковими рисами, гумор	Повторюваність ворогів, нерівномірний баланс класів	Покращення замку за золото між проходженнями
Rogue	Покрокове дослідження підземель	Процедурна генерація, простота механік, основоположник жанру	Відсутність графіки, майже нічого не пояснюється у грі	Засновник жанру roguelike, основні принципи «Берлінських критеріїв»

Основним висновком є те, що положення, сформульовані у «Берлінських критеріях», дійсно є важливими для ігор жанру roguelike. Оскільки кожне проходження гри після смерті починається з самого початку, необхідна наявність ефективних механік реіграбельності. До таких механік належать процедурна генерація рівнів, достатній рівень складності, що кидає виклик гравцю, а також обмежені ресурси, зокрема здоров'я, набої, їжа тощо. Саме ці аспекти, разом з іншими, передбаченими «Берлінськими критеріями», забезпечують гравцеві глибокий, захопливий досвід і справжнє задоволення від гри.

Основною ж проблемою більшості ігор даного жанру є незрозумілість для нового гравця жанру, а що найгірше навіть не для гравця який вже грав в ігри подібного жанру. Вона проявляється як в незрозумілості інтерфейсу деяких представників жанру, так і в відсутності деяких пояснень у грі, як то наприклад опис предметів.

Враховуючи інформацію отриману при дослідженні було прийнято рішення створити гру в жанрі Roguelike, яка буде давати грацю достане різноманіття для реіграбельності, з привабливим зрозумілим кожному інтерфейсом, та лояльну для нових граців жанру. При цьому мають бути дотримані основні принципи жанру які були сформовані у «Берлінських критеріях».

1.3. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

В якості рушія для розробки програмного продукту було обрано Unity з мовою програмування C#. Таке рішення є цілком обґрунтованим, адже Unity на сьогодні є одним із найпотужніших та найпопулярніших інструментів для створення відеоігор. Особливо добре він підходить для розробки 2D-ігор, завдяки чому ідеально вписується в концепцію обраного проекту.

Unity – це популярний кросплатформовий рушій для розробки відеоігор та інтерактивних додатків, створений компанією Unity Technologies. Він широко

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		20

використовується як в інді-розробці, так і в комерційних проєктах завдяки своїй гнучкості, зручності та потужному інструментарію.

Платформа пропонує розробникам широкий спектр можливостей: зручна робота з анімацією, реалістична фізика, гнучке налаштування інтерфейсу, підтримка візуального редагування сцен і вбудовані інструменти дебагу. Завдяки цьому процес створення гри стає не лише ефективним, але й більш інтуїтивно зрозумілим навіть для невеликої команди або індивідуального розробника.

Вся основна логіка гри реалізується вручну, що дозволяє забезпечити максимальний контроль над ігровим процесом, адаптувати поведінку персонажів, баланс системи та інші механіки під специфіку жанру рогалик. Такий підхід гарантує гнучкість у реалізації ідей, а також дозволяє легко масштабувати проєкт у майбутньому.

Плюси Unity включають зручний та зрозумілий інтерфейс, простоту інтеграції програмного коду у вже діючий рушій, а також можливість створювати додатки для різних платформ. Unity дозволяє працювати з великою кількістю ігрових фрагментів, таких як ефекти, спрайти, анімації тощо. Крім того, рушій має потужну систему фізики, яка забезпечує реалістичні рухи й взаємодії об'єктів. Важливою перевагою є також інтеграція з численними бібліотеками, що розширює можливості розробки в процесі роботи над проєктом.

Для порівняння розглянемо інший популярний рушій - Unreal Engine відзначається високою графічною потужністю, підтримкою складної 3D-графіки, реалістичного освітлення і фізики. Він часто використовується для створення великих AAA-проєктів із фокусом на якість візуалізації. Unreal Engine використовує мову C++ та має систему Blueprints для візуального скриптингу, що дає можливість створювати логіку без глибокого кодування. Однак, для 2D-ігор Unreal може бути надмірно складним і важким у налаштуванні.

У висновку можна сказати що для нашого проєкту, орієнтованого на 2D-ігру в жанрі рогалик, Unity є більш оптимальним вибором. Його простота,

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		
						21

гнучкість, зручність роботи з 2D-елементами та підтримка кросплатформеності дозволяють швидко та ефективно реалізувати задуманий функціонал. Крім того, використання C# робить розробку більш доступною навіть для невеликої команди або індивідуального розробника. Unreal Engine, хоч і потужніший у 3D-графіці, у нашому випадку буде зайвим ускладненням, що не дасть суттєвих переваг для 2D-проєкту. Тому саме Unity найкраще відповідає цілям і вимогам нашої гри.

Висновки до першого розділу.

Проведено аналіз предметної області, у результаті якого були визначені основні вимоги до гри в жанрі roguelike, зокрема: ключові ігрові механіки, особливості генерації рівнів, система розвитку персонажа та взаємодія з навколишнім світом.

В процесі дослідження були проаналізовані та порівняні аналогічні та концептуально схожі продукти в цьому жанрі, що дозволило виділити їхні сильні сторони та недоліки, а також врахувати ці аспекти при розробці власної гри.

На основі отриманих вимог і критеріїв було сформовано набір основних технологій і засобів реалізації, які будуть використані для створення проєкту. Це забезпечить відповідність гри поставленим цілям та вимогам, а також підвищить ефективність процесу розробки.

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

РОЗДІЛ 2. ПРОЕКТУВАННЯ ГРИ "ПЕКЕЛЬНИЙ КАЛІБР" У ЖАНРІ ROGUELIKE

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

Оскільки програмний продукт належить до категорії ігор, головним та єдиним актором у проєкті виступає гравець. Саме він є центральною фігурою, що безпосередньо взаємодіє з ігровою системою, ініціює ключові дії в межах геймплею та формує хід подій завдяки своїм рішенням і діям. Його активність охоплює всі основні аспекти гри — від керування персонажем до взаємодії з об'єктами навколишнього середовища.

З метою покращення розуміння структури проєкту, та для більш чіткого розуміння функціональних можливостей системи, її логіки та взаємодії гравця з ігровими механіками, було створено діаграму варіантів використання (Use Case Diagram) (рис. 2.1). Ця діаграма наочно демонструє ключові сценарії, в яких бере участь гравець, і дозволяє візуалізувати структуру можливих дій у межах програмного продукту. Кожен варіант використання представляє окрему функціональність гри, яка може бути ініційована користувачем. Вона відображає зв'язки між актором та основними функціями додатку.

Основні варіанти використання це: керування персонажем, стрільба, відкриття скринь, підбирання речей і т.д.

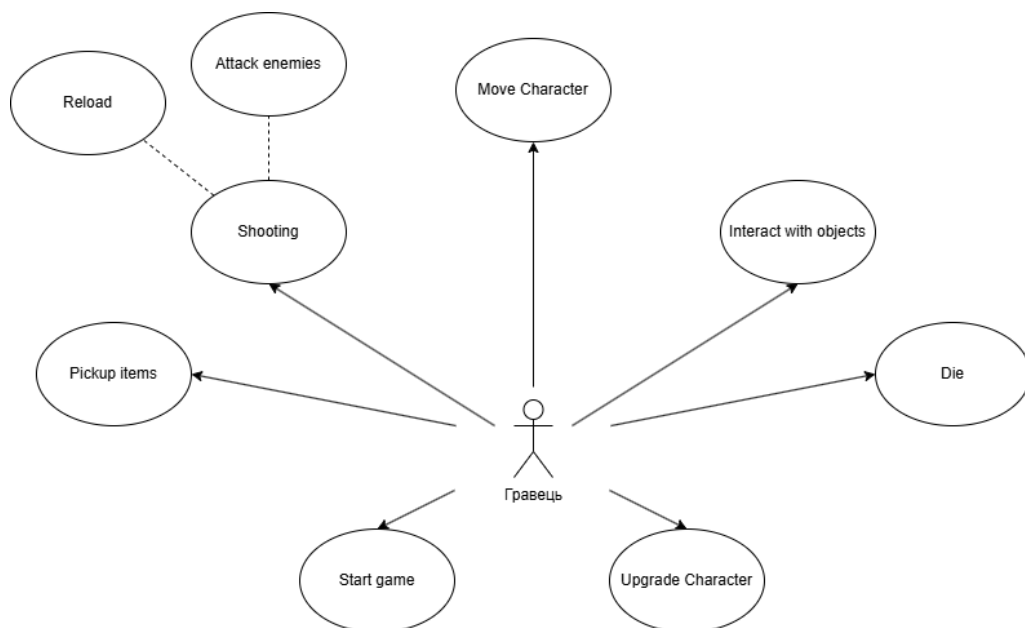


Рис. 2.1. Діаграма варіантів використання

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				23
Змн.	Арк.	№ докум.	Підпис	Дата		

Опис діаграми варіантів використання наведено в таблицях 2.1 – 2.9

Таблиця 2.1

Опис прецеденту «Початок гри»

Назва варіанту використання	Початок гри
Основні актори	Гравець
Короткий опис	Спавн гравця на локації.

Таблиця 2.2

Опис прецеденту «Прокачка персонажу»

Назва варіанту використання	Прокачка персонажу
Основні актори	Гравець
Короткий опис	Прокачка певних статів гравця, таких як швидкість перезарядки, кількість набойів, кількість поінтів здоров'я та інші.

Таблиця 2.3

Опис прецеденту «Смерть персонажу»

Назва варіанту використання	Смерть персонажу
Основні актори	Гравець
Короткий опис	Закінчення поточного забігу по локації(сесії гри), спричинене смертю персонажа.

Таблиця 2.4

Опис прецеденту «Взаємодія з об'єктами»

Назва варіанту використання	Взаємодія з об'єктами
Основні актори	Гравець
Короткий опис	Взаємодія гравця з об'єктами, наприклад відкриття скринь, бочок і подібного.

Таблиця 2.5

Опис прецеденту «Рух персонажу»

Назва варіанту використання	Рух персонажу
Основні актори	Гравець

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк. 24
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

Продовження таблиці 2.5

Короткий опис	Пересування персонажу по мапі.
---------------	--------------------------------

Таблиця 2.6

Опис прецеденту «Стрільба»

Назва варіанту використання	Стрільба
Основні актори	Гравець
Короткий опис	Персонаж випускає кулі з поточної зброї.

Таблиця 2.7

Опис прецеденту «Нанесення шкоди противникам»

Назва варіанту використання	Нанесення шкоди противникам
Основні актори	Гравець
Короткий опис	Кулі випущені персонажем, що влучають в противників, наносять їм шкоду.

Таблиця 2.8

Опис прецеденту «Перезарядка»

Назва варіанту використання	Перезарядка
Основні актори	Гравець
Короткий опис	При закінченні куль в гравця активується перезарядка. Або ж коли гравець розуміючи що на наступну сутичку куль не вистачить, сам може ініціювати перезарядку.

Таблиця 2.9

Опис прецеденту «Підняття предметів»

Назва варіанту використання	Підняття предметів
Основні актори	Гравець
Короткий опис	При закінченні куль в гравця активується перезарядка. Або ж коли гравець розуміючи що на наступну сутичку куль не вистачить, сам може ініціювати перезарядку.

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				25
Змн.	Арк.	№ докум.	Підпис	Дата		

Основна ціль розробки гри «Пекельний калібр» в жанрі roguelike є створення геймплейно захопливої та візуально привабливої гри, орієнтованої на гравців з будь-яким рівнем досвіду та навичок. Проект має бути достатньо лояльним у плані складності та зрозумілим для користувача, незалежно від його досвіду в іграх загалом. Окрім забезпечення якісного користувацького досвіду, важливою метою є створення технічно гнучкого та масштабованого програмного продукту, який легко адаптується до подальших оновлень, доповнень і вдосконалень.

Отже – метою розробки гри «Пекельний калібр» у жанрі roguelike є створення захопливої, технологічно якісної та доступної гри, у sci-fi сетингу, яка поєднує елементи випадковості, динамічного бою та стратегічного планування. Проект спрямований на формування унікального ігрового досвіду для широкого кола користувачів — як новачків, так і досвідчених гравців. Особливу увагу приділено забезпеченню балансу між складністю та зрозумілістю, а також побудові архітектури гри, яка дозволить ефективно розширювати й оновлювати функціонал у майбутньому. У підсумку, мета полягає не лише у створенні розважального продукту, але й у розробці технічно досконалої основи для подальшого розвитку ігрового проекту.

Вимоги користувача до гри включають зручний та інтуїтивно зрозумілий інтерфейс, який дозволяє легко орієнтуватися в меню, керуванні персонажем та виборі опцій. Гра повинна мати систему процедурної генерації, що забезпечує реіграбельність завдяки випадковій генерації карти для кожного проходження. Важливо, щоб ігровий процес був динамічним та захоплюючим, з елементами випадковості та стратегії, що робить кожен рівень унікальним. Система балансу складності має забезпечувати комфортну гру як для новачків, так і для досвідчених гравців. Візуальна складова повинна відповідати стилю sci-fi та бути якісною, з приємною графікою та анімаціями. Гравець має отримувати можливість прогресу, що стимулює подальше проходження. Також важливо, щоб проєкт підтримував масштабованість і міг отримувати оновлення з новими

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

функціями, рівнями, персонажами та предметами. Керування в грі має бути зручним. При цьому має бути забезпечена взаємодія з навколишнім світом та іншими гравцями, якщо такі режими будуть передбачені.

Функціональні вимоги до гри передбачають реалізацію декількох ключових елементів. Гра повинна автоматично генерувати нові рівні (карти) з випадковими елементами для кожного проходження, що забезпечує унікальний досвід у кожній грі. Гравець має можливість керувати персонажем — рухатися, атакувати та взаємодіяти з об'єктами — за допомогою стандартних засобів управління, таких як клавіатура, мишка або контролер.

Система бою повинна включати різноманітні механіки, зокрема атаки, використання предметів і стратегічне планування під час боїв з ворогами. Прогрес персонажа передбачає можливість підвищення рівня та отримання нових покращень. Гравець також може збирати предмети, такі як зброя чи медикаменти, які впливають на характеристики персонажа або допомагають у проходженні гри.

У грі мають бути реалізовані візуальні ефекти для атак, здібностей, анімацій ворогів та інших важливих подій, що створюватиме враження динамічності. Кожен рівень або частина гри повинна мати різні шляхи для проходження, надаючи гравцю вибір та підвищуючи реіграбельність. Також передбачено можливість збереження прогресу, що дозволяє гравцю повернутися до гри в будь-який момент, зберігаючи всі досягнення та зміни.

Не функціональні вимоги включають такі аспекти. По сприйняттю: час, необхідний для освоєння основних моментів гри, має становити від 15 до 45 хвилин, а для досвідчених користувачів — менше 15 хвилин. Відповідь гри на дії користувача повинна бути миттєвою, а час завантаження залежати від потужності комп'ютера. Інтерфейс має бути інтуїтивно зрозумілим і зручним, не вимагаючи додаткової підготовки користувача. Щодо надійності, гра повинна працювати без збоїв протягом тривалого часу з мінімальним ризиком втрати прогресу чи даних. У разі аварійного завершення передбачено відновлення

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		
						27

останнього збереженого прогресу або його автоматичне збереження в реальному часі. Гра також має надавати чіткі повідомлення про помилки користувачеві, коли це необхідно. В частині продуктивності, гра повинна ефективно використовувати ресурси комп'ютера, не перевантажуючи процесор чи пам'ять, забезпечуючи плавний ігровий процес на різних типах апаратного забезпечення. При процедурній генерації рівнів та обробці великої кількості даних система має підтримувати високу продуктивність, не уповільнюючи гру.

Системні вимоги до гри передбачають певні параметри середовища виконання. Платформа гри повинна відповідати мінімальним технічним характеристикам пристроїв. Гра підтримується на операційних системах Windows 7 (64-bit) або новіших версіях, macOS 10.12 Sierra або новіших, а також на Linux із підтримкою OpenGL.

Щодо процесора, мінімальною вимогою є Intel Core i3 або AMD Ryzen 3, проте для кращої продуктивності рекомендується Intel Core i5 або AMD Ryzen 5. Оперативна пам'ять повинна становити щонайменше 4 GB, а рекомендовано — 8 GB. Відеокарта повинна бути не нижче рівня NVIDIA GeForce GT 630 або AMD Radeon HD 6570, а для більш комфортного геймплею бажано мати NVIDIA GeForce GTX 750 або AMD Radeon RX 460 чи еквівалент.

Для встановлення гри потрібно мінімум 5 GB вільного місця на жорсткому диску, а рекомендовано мати 10 GB. Також гра повинна працювати в графічному середовищі з підтримкою OpenGL для всіх трьох основних операційних систем: Windows, macOS та Linux.

Аналіз функціональних вимог дозволив виділити наступні сутності, що забезпечать програмну реалізацію додатку. В подальшому буде показано декілька діаграм класів додатку, які будуть відображати основні елементи гри.

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

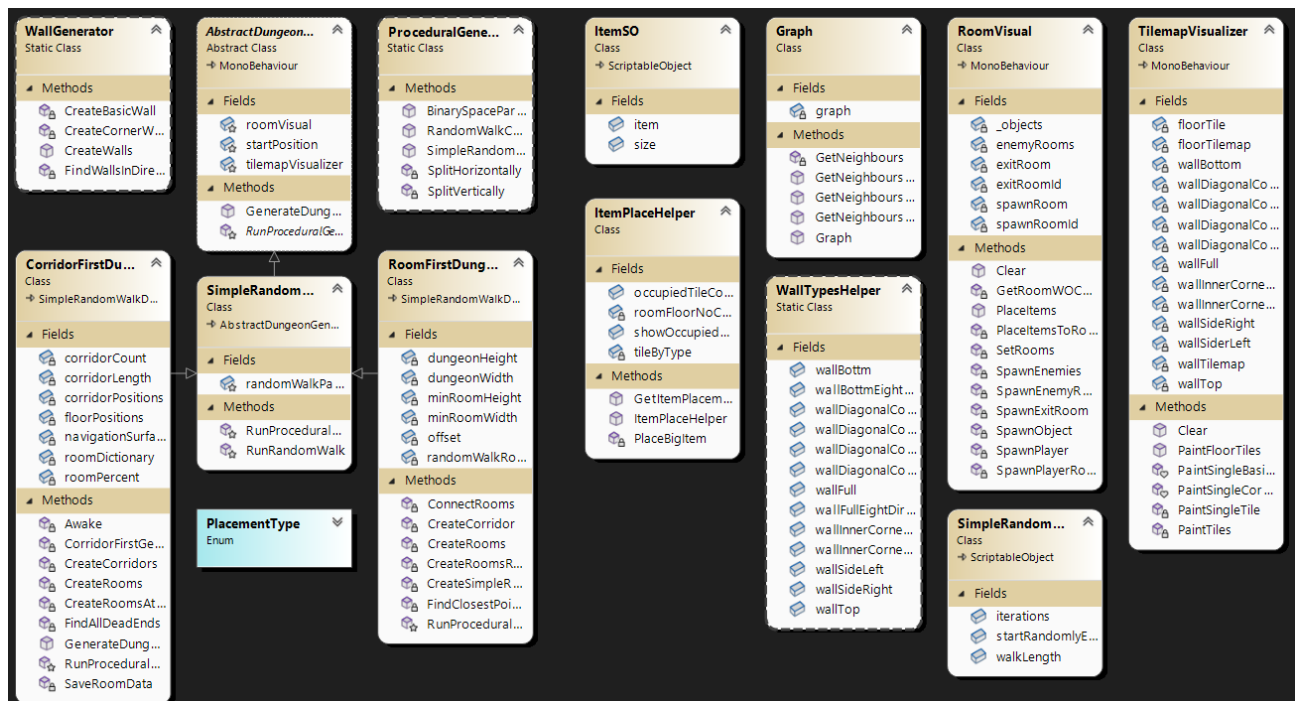


Рис. 2.2. Діаграма класів Процедурної генерації

Генерація структури підземелля організована за допомогою абстрактного класу `AbstractDungeonGenerator`, який визначає загальний інтерфейс для створення підземель. Різні алгоритми генерації реалізовані у конкретних класах, таких як `SimpleRandomDungeonGenerator`, `RoomFirstDungeonGenerator` та `CorridorFirstDungeonGenerator`. Для пошуку сусідніх клітинок на мапі використовується допоміжний клас `Graph`, що допомагає у подальшому промальовуванні підземелля. Статичний клас `ProceduralGenerationAlgorithms` містить набір базових алгоритмів, які застосовуються різними типами генерації для побудови карти.

Візуалізація мапи здійснюється класом `TilemapVisualizer`, який відповідає за відображення згенерованого плану на ігровій мапі за допомогою тайлів. Він зберігає інформацію про типи плиток, такі як підлога і стіни, та має методи для їхнього малювання. Статичні класи `WallGenerator` і `WallTypesHelper` допомагають визначати та створювати стіни на основі згенерованого плану, враховуючи різні типи з'єднань між плитками.

Розміщення ігрових об'єктів відбувається за допомогою класу `ItemPlaceHelper`, який враховує зайняті місця та типи поверхонь для коректного

розміщення предметів. Клас RoomVisual відповідає за розміщення об'єктів усередині окремих кімнат, а також за появу ворогів і гравця в підземеллі. Перерахування PlacementType визначає різні типи розміщення, наприклад, у вільному просторі або біля стін.

Для керування даними про ігрові об'єкти використовується клас ItemSO, який є ScriptableObject і зберігає інформацію про предмети, такі як назва і розмір. Також є клас SimpleRandomWalkSO, який зберігає параметри для алгоритму процедурної генерації.

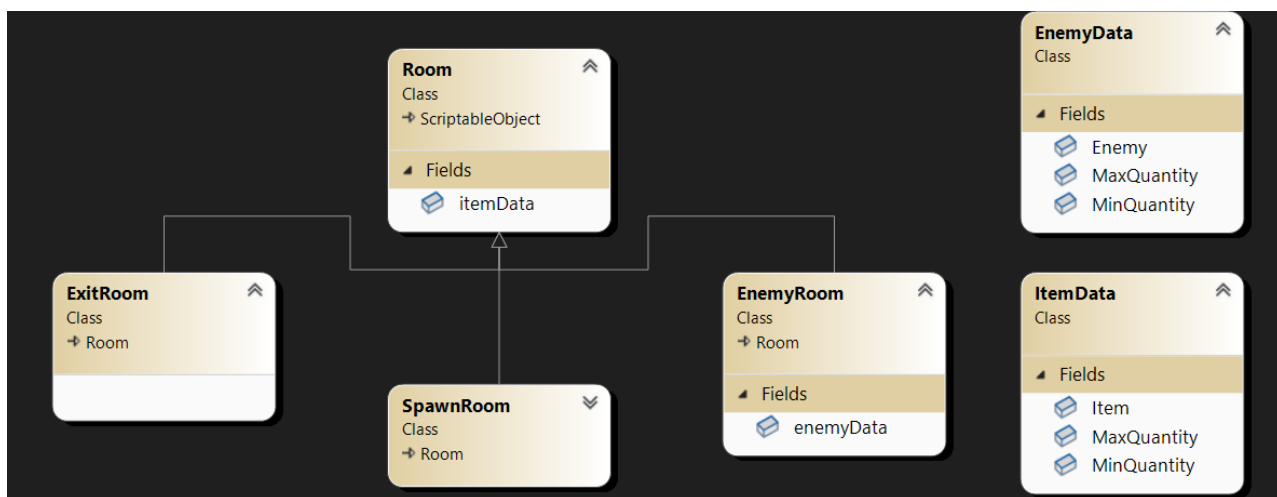


Рис. 2.3. Діаграма класів для типів кімнат

Room – Батьківський клас зберігає в собі інформацію про об'єкти які мають бути розміщені в кімнаті.

EnemyRoom, ExitRoom та SpawnRoom відповідають за типи кімнат з відповідним наповненням, кімната з ворогами це звичайний тип кімнати в якому проходить більша частина геймплею, кімната спавну це кімната в якій з'являється гравець, та з якої починається його шлях, кімната виходу позначає вихід з поточного підземелля.

ItemData та EnemyData в свою чергу відповідають за зберігання інформації про те який противник або об'єкт буде розміщено та якій кількості.

Далі розглянемо клас що відповідає за ворога, та зберігає його поточне здоров'я, максимальне здоров'я, шкоду, та декілька допоміжних елементів – це змінні які відповідають за отримання шкоди противником.

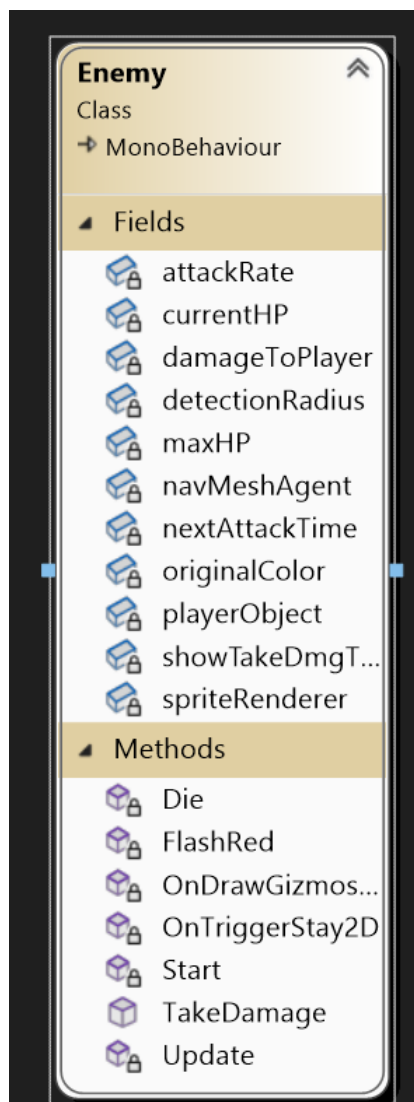


Рис. 2.4. Клас противника

Також клас містить методи отримання шкоди та смерті, та метод візуалізації шкоди нанесеної противнику.

У процесі розробки гри було створено початкові версії основних класів, які відповідають за поведінку гравця, зброї та боєприпасів (рисунок 2.5.). Нижче подано опис їхньої структури, проте слід зауважити, що це не фінальні версії класів — вони можуть бути розширені або змінені відповідно до потреб проєкту.

Клас Player містить поля, що відповідають за найближчий об'єкт для взаємодії, поточну зброю гравця, радіус взаємодії, швидкість руху, компонент Rigidbody, об'єкт зброї на сцені, позицію миші та назву шару взаємодії. Серед основних методів — пошук об'єкта для взаємодії, визначення напрямку руху, взаємодія з об'єктами та реалізація самого руху.

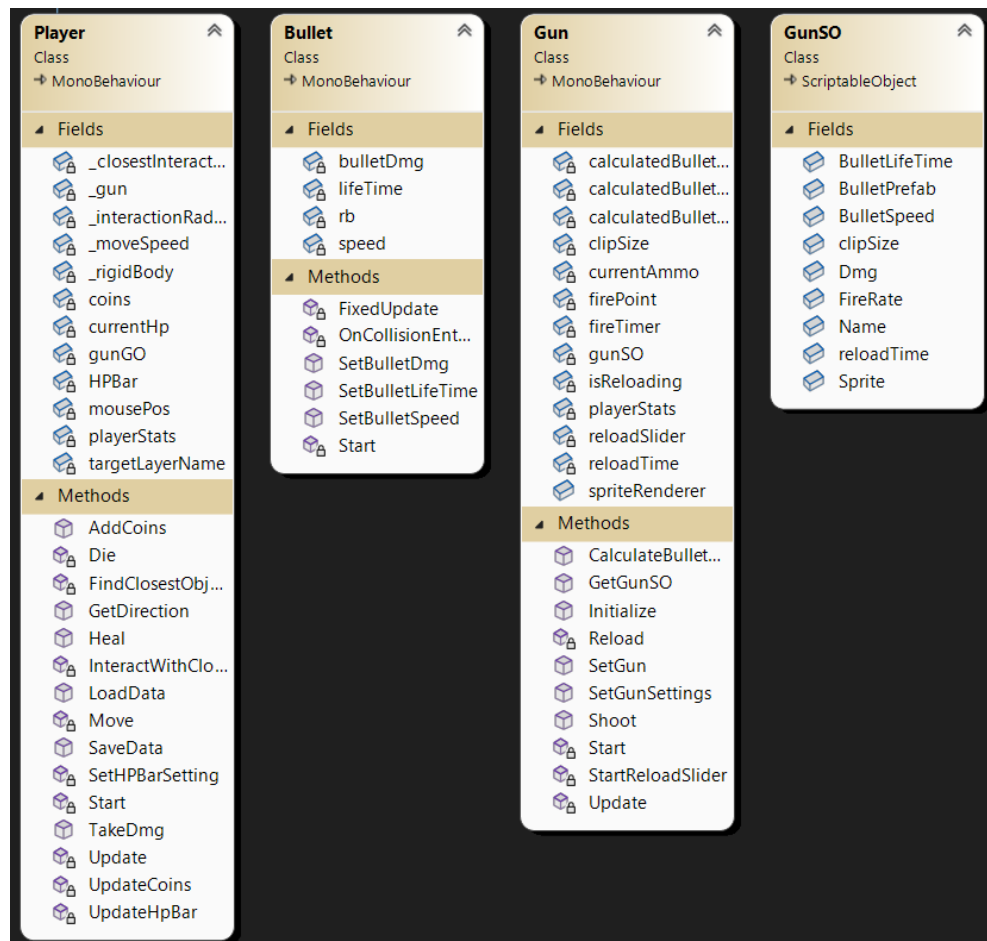


Рис. 2.5. Клас гравця та його зброї

Gun відповідає за функціональність зброї. До його полів належать точка вильоту кулі, таймер пострілів, об'єкт налаштувань зброї типу GunSO та компонент SpriteRenderer. Методами класу є отримання налаштувань зброї, встановлення параметрів кулі, налаштування зброї та реалізація пострілу.

Клас Bullet містить параметри життя кулі, її швидкість та Rigidbody2D компонент для фізичного контролю. Він має методи для встановлення часу життя та швидкості кулі.

Клас GunSO є ScriptableObject і служить для зберігання налаштувань зброї, таких як тривалість життя кулі, швидкість, шкода, швидкострільність, назва та спрайт зброї.

Ці класи закладають фундамент логіки стрільби та взаємодії, проте їх реалізація буде вдосконалюватися протягом розробки з урахуванням балансу, нових механік та оптимізації.

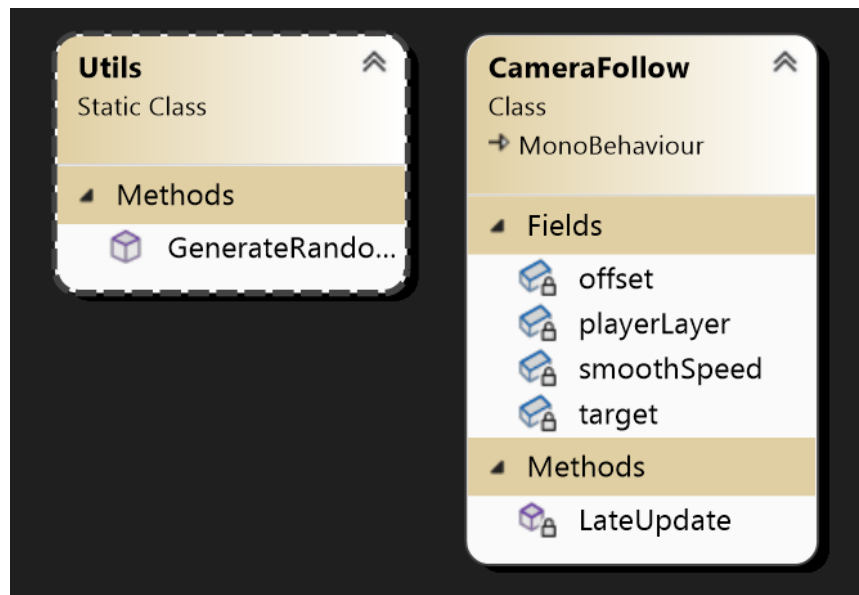


Рис. 2.6. Класи Утиліт та Слідування камери

Utils – має містити методи які будуть використовуватись глобально в різних сегментах програмного кода, по типу генерації випадкового числа.

CameraFollow – клас який змушує камеру слідувати за гравцем.

Також в програмному продукті містяться класи які відповідають за речі які лежать на підлозі, і з якими може взаємодіяти гравець. Вони є нащадками класу Interactable, який містить одноіменний метод який вони перевизначають.

Перейдемо до прикладу цього класу а саме – Клас DroppedGun відповідає за поведінку зброї, яка була скинута на землю і з якою може взаємодіяти гравець. У його структурі передбачено кілька основних полів: gunSO — посилання на об'єкт типу GunSO, що містить дані про конкретну зброю; rotationSpeed — швидкість обертання об'єкта на підлозі для надання йому візуальної привабливості; spriteRenderer — компонент для відображення зовнішнього вигляду зброї на сцені.

Серед основних методів класу варто виділити FindPlayerGunLayerObject(), який шукає зброю гравця на сцені; Interact(Transform interactorTransform) — перевизначений метод взаємодії, що активується при взаємодії гравця з предметом; SetGunSO(GunSO gunData) — метод для встановлення налаштувань зброї; та SetPlayerGun(GameObject player) — метод, який відповідає за передачу зброї гравцю після підбору.

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

Цей клас дозволяє реалізувати механіку підбору зброї, що є важливою складовою геймплею в грі жанру roguelike.

Також до нащадків класу Interactable належать класи Chest який відповідає за скриню з якої гравець буде отримувати предмети, та клас Portal за допомогою якого гравець буде переходити на інші рівні.

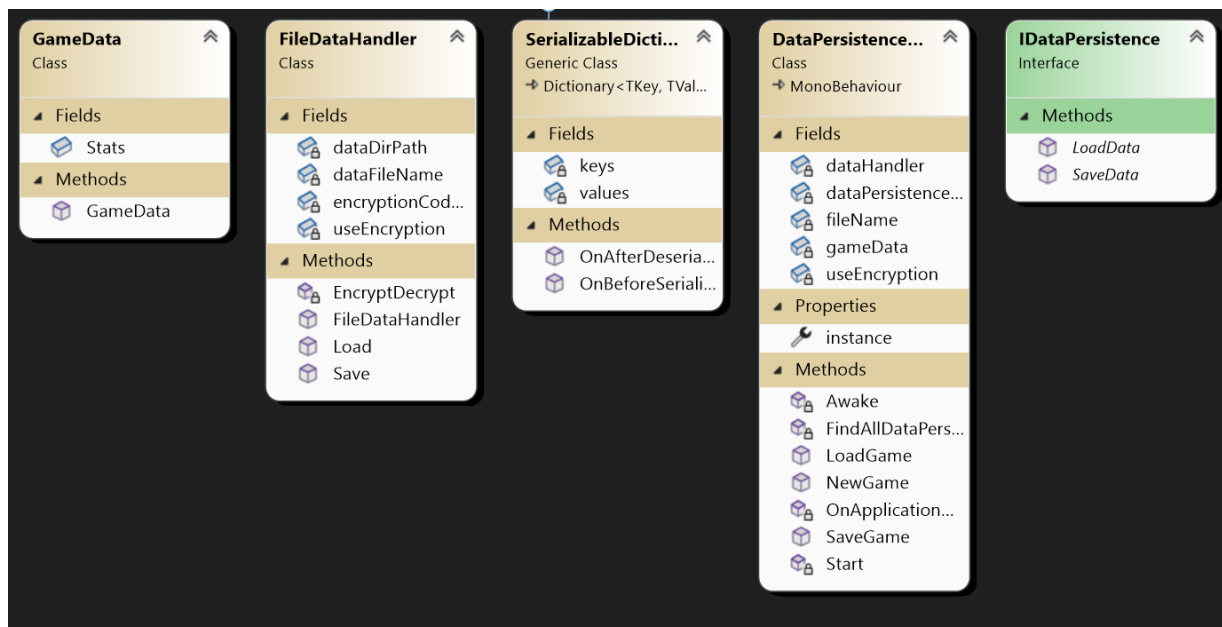


Рис. 2.7. Класи що відповідають за збереження

У системі збереження гри використовується кілька ключових компонентів, кожен з яких виконує свою роль. Клас GameData виступає в ролі контейнера для всіх даних про стан гри, зокрема даних про гравця, його прокачку, спорядження та інші параметри, які мають бути збережені.

Для серіалізації складних структур, зокрема словників, у Unity використовується SerializableDictionary у поєднанні з інтерфейсом ISerializationCallbackReceiver, що дозволяє обійти обмеження Unity щодо серіалізації типів Dictionary.

Клас DataPersistenceManager є головним керуючим компонентом, реалізованим як Singleton, що відповідає за централізоване керування збереженням та завантаженням даних. При старті гри цей менеджер створює або ініціалізує об'єкт FileDataHandler, завантажує наявні ігрові дані або створює нову гру, якщо даних немає. Під час завантаження менеджер знаходить усі об'єкти

сцени, які реалізують інтерфейс IDataPersistence, і передає їм відповідні дані для відновлення стану. При збереженні менеджер знову звертається до цих об'єктів, щоб зібрати актуальні дані і зберегти їх у файл.

Інтерфейс IDataPersistence забезпечує гнучкий механізм підключення будь-якого об'єкта до системи збереження. Класи, які реалізують цей інтерфейс, повинні мати методи LoadData(GameData data) для завантаження стану та SaveData(ref GameData data) для його збереження.

Нарешті, клас FileDataHandler відповідає за роботу з файловою системою: він здійснює читання з файлу, запис даних у файл, а також, за потреби, шифрування і розшифрування збережених даних.

Уся система працює таким чином, що при запуску гри відбувається завантаження збережених даних і відновлення стану відповідних об'єктів. У процесі гри, або перед її завершенням, дані знову збираються і зберігаються, забезпечуючи безперервність прогресу гравця при кожному наступному запуску.



Рис. 2.8. Повна діаграма класів

2.2. Розробка системи збереження даних

Розроблена система збереження даних базується на чіткій архітектурі, що забезпечує надійне та гнучке керування станом гри. Ключовим елементом є клас `GameData`, який слугує централізованим контейнером для всіх ігрових даних, що потребують збереження.

Дані для збереження це в першу чергу: прокачка гравця.

Для забезпечення можливості серіалізації складних структур даних, зокрема словників, які широко використовуються для відстеження стану ігрових об'єктів за їхніми унікальними ідентифікаторами, було застосовано спеціалізований клас `SerializableDictionary`. Цей клас реалізує інтерфейс `ISerializationCallbackReceiver`, надаючи механізм для перетворення словника в серіалізовані списки ключів та значень перед збереженням та відновлення словника з цих списків після завантаження.

Основну логіку керування процесом збереження та завантаження інкапсульовано в класі `DataPersistenceManager`. Цей клас виступає в ролі координатора, відповідаючи за ініціалізацію системи, завантаження існуючих даних або створення нових при першому запуску, а також за ініціювання процесу збереження за запитом або при завершенні роботи програми. `DataPersistenceManager` використовує патерн `Singleton`, гарантуючи наявність лише одного екземпляра в грі, що спрощує доступ до його функціональності з різних частин кодової бази.

За фактичну роботу з файловою системою відповідає клас `FileDataHandler`. Він інкапсулює логіку читання та запису даних у файл, підтримуючи можливість використання шифрування для захисту збережених даних від несанкціонованого доступу.

Зв'язок між різними ігровими компонентами та системою збереження здійснюється через інтерфейс `IDataPersistence`. Будь-який клас, чий стан необхідно зберігати, повинен реалізувати цей інтерфейс, надаючи власну логіку для завантаження своїх даних з об'єкта `GameData` та для збереження свого поточного стану в цей об'єкт. `DataPersistenceManager` відповідає за пошук усіх

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

реалізацій IDataPersistence у сцені та виклик їхніх методів LoadData та SaveData у відповідні моменти.

Таким чином, розроблена система збереження даних характеризується модульністю та чітким розподілом відповідальності. Клас GameData виступає як централізоване сховище, SerializableDictionary забезпечує підтримку серіалізації словників, DataPersistenceManager координує процес збереження та завантаження, FileDataHandler обробляє файлові операції, а інтерфейс IDataPersistence надає стандартний спосіб для ігрових компонентів взаємодіяти з системою збереження. Ця архітектура спрощує додавання нових даних для збереження та забезпечує легкість у підтримці та розширенні системи в майбутньому.

2.3. Проектування та реалізація алгоритмів роботи системи

Додаток є простим у використанні, має всі основні функції гри цього жанру, що дозволяє користувачам легко взаємодіяти з усіми елементами гри. Алгоритм роботи з платформою зображено на рисунку 2.9, також описано детально його принцип.

Після запуску системи користувач потрапляє до головного меню, де йому надаються основні опції для взаємодії з грою. Серед них є можливість почати нову гру або вийти з гри. Якщо користувач обирає вихід, система завершує свою роботу. У разі вибору початку гри, система перевіряє, чи існують збережені дані. Якщо збереження є, користувачеві пропонується завантажити існуючу гру. Якщо збережень немає, створюється нове.

Після цього користувач потрапляє до меню покращень, де він може налаштувати або покращити характеристики свого персонажа. Завершивши дії в цьому меню, він переходить безпосередньо до початку ігрового процесу.

Перед початком гри система процедурно генерує нову мапу підземелля. Далі розпочинається основна частина гри, в якій користувач взаємодіє з ігровим світом, досліджує підземелля, бореться з ворогами, збирає предмети та виконує інші дії.

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

Під час гри можуть трапитись дві ключові події. Якщо гравець гине, система повертає його до меню покращень, звідки він може знову продовжити гру або повернутися до головного меню. Якщо ж гравець знаходить і проходить вихід з підземелля, система запускає повторну генерацію нової мапи і гра продовжується з новими умовами.

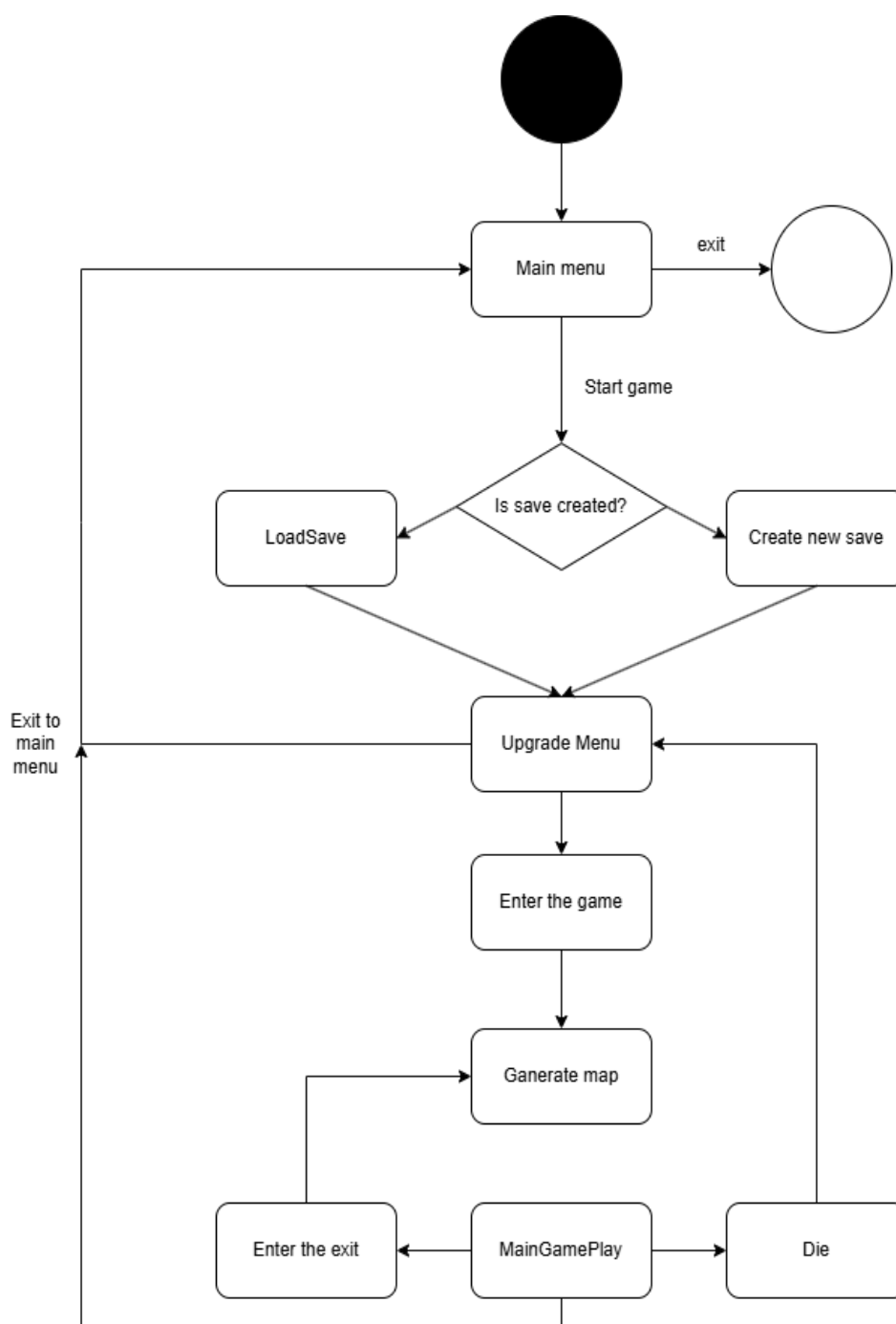


Рис. 2.9. Діаграма активності

2.4. Реалізація функціоналу гри «Пекельний кал» в жанрі Roguelike

Розглянемо основні етапи розробки гри. Почнемо з процедурної генерації, батьківським класом для всієї генерації в грі є `AbstractDungeonGenerator`. Основний метод який в ньому міститься це `RunProceduralGeneration` – це абстрактний метод який перевизначають класи що наслідують його, визначачи свій метод генерації.

Наступним важливим класом в генерації локації є `ProceduralGenerationAlgorithms` – він відповідає за самі алгоритми генерації.

Наступним за ієрархією важливості є клас `CorridorFirstDungeonGenerator` (або ж `roomFirst` що означає, що спочатку буде відбуватись генерація самих кімнат), він використовує вище наведені алгоритми генерації для створення локації. Сам клас та його методи можна оглянути в додатку А. Основним його методом є `CreateCorridors` – в ході виконання від заповнює позиції потенційного місця кімнат та позиції підлоги, після чого запускає генерацію кімнат, наведена в методі `CreateRooms`, самі ці методи викликаються в іншому методі `CorridorFirstGeneration` Який власне і запускає всю генерацію локації. В ньому можна побачити метод `FindAllDeadEnds` та `CreateRoomsAtDeadEnd` які відповідають за заповнення глухих кутів в коридорах. Також після цього всього можна прослідкувати запуск промальовування підлоги та стін, перейдемо до них.

За це відповідає клас `TilemapVisualizer` та `WallGenerator`, перейдемо до розгляду першого з них – `TilemapVisualizer` він має інформацію про всі тайли які мають бути відображені в грі та промальовує їх отримуючи інформацію про те який яме тайл має бути на цьому місці.

Поки перейдемо до класу `WallGenerator` - він аналізує позиції підлоги та переглядає всіх сусідів певного тайлу підлоги. Повертаючись до того як система розуміє який саме тайл промальовувати, вона переглядає кожен з 4-х (для базових стін) або 8-ми (для кутів) фіксованих напрямків навколо позиції потенційної стіни. Для кожного напрямку перевіряється, чи існує підлога в цій сусідній клітинці, і якщо так — до бінарного рядка додається '1', інакше — '0'. Таким чином формується бітовий шаблон, який відповідає конфігурації сусідів.

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				39
Змн.	Арк.	№ докум.	Підпис	Дата		

Потім цей шаблон перетворюється на число й порівнюється з наборами у відповідних множинах (`HashSet<int>`) з класу `WallTypesHelper`, що дозволяє визначити тип стіни (верхня, бокова, внутрішній або діагональний кут тощо) та промальовувати правильний тайл. Всі бінарні значення в `WallTypesHelper`, що відповідають за те який куди тайл поставити, перевірені практично і загалом не потребують доопрацювання, але за потреби, якщо в грі знайдено тайл який не має бути на своєму місці, можна порахувати його Хешсет та внести відповідні правки до хелпера, щоб в подальшому цей тайл відмальовувався правильно.

Сам клас `WallTypesHelper` можна переглянути в додатку В.

Після цього всього кімнати заповнюються предметами, ворогами та в одній з них, обраний випадковим чином, з'являється гравець, також випадковим чином обирається кімната з порталом яка дозволить гравцю перейти на інший рівень.

Сам процес активується в класі `CorridorFirstDG`, а відповідає за нього клас `RoomVisual`, сам клас детально можна буде переглянути в додатку С.

Допоміжним класом для цього є `ItemPlaceHelper`, який рахує розмір предмету та чи поміститься він в певне місце.

Клас `RoomVisual` та `ItemPlaceHelper` можна переглянути в додатку С.

Гравець у грі керується методом руху, який змінює його швидкість залежно від напрямку, заданого введенням з клавіатури. Напрямок визначається окремим методом, що враховує натискання клавіш WASD і повертає нормалізований вектор, який і задає рух. У результаті гравець може вільно переміщуватись по мапі.

Окрім руху, гравець має можливість взаємодіяти з об'єктами навколо. Для цього використовується механізм пошуку найближчого об'єкта з потрібного шару в певному радіусі. Цей об'єкт зберігається у спеціальному полі. Якщо гравець натискає клавішу взаємодії, викликається метод, який перевіряє, чи є знайдений об'єкт взаємодійним, і якщо так — виконується взаємодія (наприклад, підняття предмета).

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		40

Гравець також може отримувати шкоду. Метод отримання шкоди зменшує поточне здоров'я гравця, а метод смерті спрацьовує, коли його здоров'я зменшується до нуля або нижче. Це призведе до повернення в меню покращень.

Також у гравця є метод отримання коштів. Весь код класу можна переглянути в додатках (Додаток Е).

Зброя гравця складається з двох частин — самої зброї та кулі. Клас кулі відповідає за її рух і має параметри швидкості, часу життя та шкоди. Кожен кадр вона рухається вперед, а якщо стикається з ворогом, той отримує шкоду, після чого куля знищується.

Зброя має основну функцію — стрільбу. Після кожного пострілу запускається таймер, який контролює швидкострільність. Вогонь відкривається лише тоді, коли таймер завершено, що дозволяє реалізувати індивідуальні параметри кожного виду зброї.

Вороги у грі реалізовані через окремий клас. Кожен ворог має здоров'я та можливість реагувати на отримання шкоди. Коли ворог зазнає ушкоджень, його спрайт короткочасно змінює колір на червоний, щоб показати візуальний ефект шкоди. Якщо здоров'я зменшується до нуля, ворог гине — його об'єкт видаляється зі сцени.

Таким чином, реалізовано базову логіку руху гравця, взаємодії з навколишніми об'єктами, стрільби, обробки шкоди як для гравця, так і для ворогів, що забезпечує основний ігровий процес.

За збереження даних у грі відповідає низка спеціалізованих класів. Основою є інтерфейс `IDataPersistence`, який задає базові методи для збереження та завантаження даних — кожен об'єкт, що має свій стан, реалізує цей інтерфейс, щоб мати змогу інтегруватися у загальну систему збереження.

Для роботи з файловою системою використовується клас `FileDataHandler`. Він відповідає за читання збережених даних із файлу та запис нових даних у файл. Крім того, підтримується просте шифрування та дешифрування вмісту для додаткового захисту інформації.

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

Ключовим компонентом є `DataPersistenceManager`. Це менеджер, що координує увесь процес збереження та завантаження. Під час запуску гри він завантажує дані за допомогою `FileDataHandler` і передає їх усім відповідним об'єктам. При збереженні — навпаки, отримує дані від усіх об'єктів і зберігає їх у файл.

Всі ігрові дані об'єднані в серіалізований клас `GameData`, який містить інформацію про стан гравця, інвентар, спорядження, розташування персонажа, а також стан ворожих об'єктів, дверей та іншого важливого контенту.

Оскільки Unity за замовчуванням не дозволяє серіалізувати словники, для збереження колекцій типу `Dictionary` використовується клас `SerializableDictionary<TKey, TValue>`, який адаптований під систему серіалізації Unity.

Висновки до другого розділу

В процесі проектування гри «Пекельний калібр» в жанрі roguelike були проаналізовані усі варіанти використання програмного продукту відповідно до вимог програмного забезпечення. На основі отриманого результату була спроектована структура гри. Проектування включало в себе моделювання структури проекту та взаємодії між їх складовими частинами.

Також було спроектовано систему для збереження даних у грі, такий як наприклад прокачка персонажу.

Було описано алгоритм роботи з грою та описані послідовні пункти варіантів його використання. Також були наведені основні класи та методи які були реалізовані для роботи додатку.

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		42

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ ДОДАТКУ

3.1 Порядок встановлення та налаштування системи

Гра створена за допомогою ігрового рушія Unity і призначена для запуску на операційній системі Windows. Щоб почати користуватися додатком, достатньо завантажити готову збірку гри, яка надається у вигляді архіву або окремої папки з усіма необхідними файлами. Після завантаження архів слід розпакувати у зручне місце на комп'ютері, після чого користувач може запустити гру через виконуваний файл з розширенням .exe, наприклад HellCaliber.exe. Гра не потребує додаткової інсталяції або встановлення сторонніх компонентів, оскільки всі необхідні бібліотеки вже включені до складу збірки. При першому запуску гра автоматично створює усі необхідні системні файли, включаючи каталог для збереження прогресу. Працює вона повністю автономно, не вимагає підключення до інтернету і забезпечує стабільну роботу на більшості сучасних ПК під управлінням Windows.

3.2 Структура інтерфейсу та порядок взаємодії з грою

Продемонструємо інтерфейс системи на прикладі основного потоку використання. При запуску гри користувач потрапляє в головне меню де його зустрічають дві кнопки – почати гру та вийти з гри.

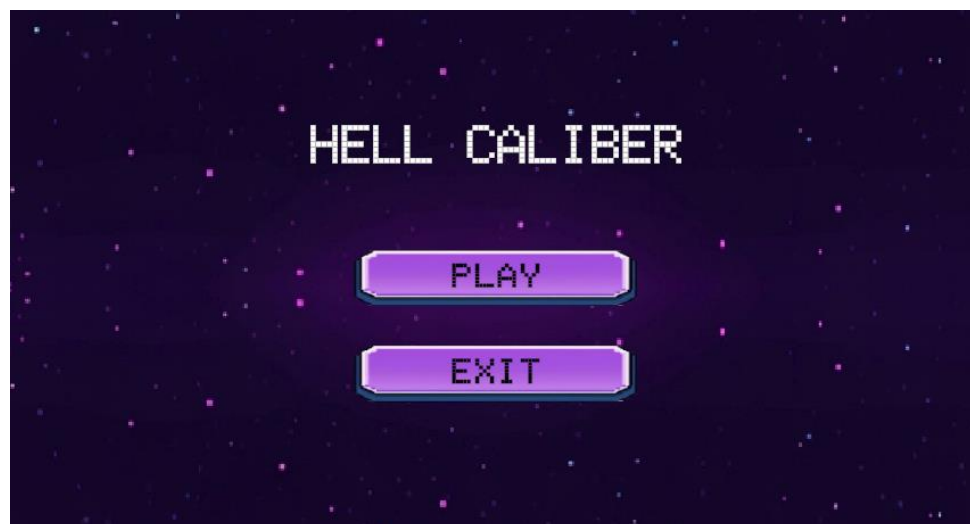


Рис. 3.1 Головне меню

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

Якщо користувач натисне на кнопку «Exit» - гра завершиться, якщо ж на плей його перенесе до вікна покращень гравця, наведену на рисунку 3.2.

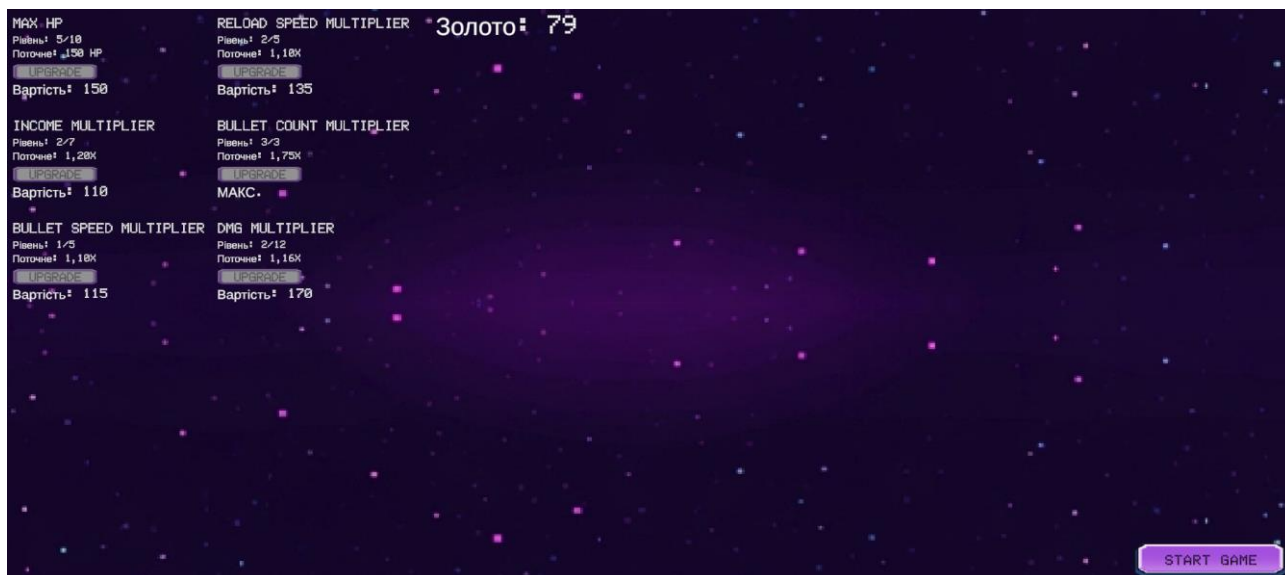


Рис. 3.2 Сторінка покращень гравця

На екрані покращень гравця представлено декілька основних характеристик, які можна покращувати: максимальний запас здоров'я (MAX HP), множник швидкості перезарядки (RELOAD SPEED MULTIPLIER), множник доходу (INCOME MULTIPLIER), множник кількості куль (BULLET COUNT MULTIPLIER), множник швидкості польоту куль (BULLET SPEED MULTIPLIER) та множник завданої шкоди (DMG MULTIPLIER), у кожного з них можна побачити ціну для покращення, поточний та максимальний рівні, та значення множників.

Також на сторінці можна побачити поточну кількість валюти у гравця, та кнопку переходу до гри.

Перейдемо до самого візуалу ігрового процесу, коли ми натискаємо почати гру, нас переносить в випадково згенеровану локацію, з випадковими предметами, ворогами і т. д.

На рисунку 3.3 можна детальніше ознайомитись з інтерфейсом та візуалом гри.

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		44



Рис. 3.3 Ігровий процес

На рисунку помітно що в центрі екрану знаходиться сам ігровий персонаж, керування яким здійснюється клавішами WASD, а стрільба реалізована за допомогою мишки.

В правому нижньому куті можна побачити поточну кількість грошей на балансі гравця, а в лівому верхньому його поточне здоров'я.



Рис. 3.4 Постріл

При натисканні на ліву кнопку миші, відбудеться постріл, в ту сторону де гравець тримає свій курсор, в момент натискання із зброї гравця вилетить куля, показана на рисунку 3.4, якщо куля зустрінить з перешкодою, вона вріжиться в

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		45

неї і зникне, якщо ця перешкода – противник, він отримає шкоду, таку яку може заподіяти поточна зброя гравця.

Кожна зброя має свою кількість набоїв, і час від часу її треба перезаряджати (за це відповідає кнопка «R»), якщо ж набоїв в поточному магазині залишиться 0, зброя автоматично піде на перезарядку, візуальне відображення перезарядження показано на рисунку 3.5.



Рис. 3.5 Перезарядка

В ході гри гравець буде зустрічатись з ворогами, які будуть очікувати його в деяких кімнатах на локації, кількість та розміщення ворогів, як і кімнат, випадкове. Ворогі отримують шкоду від пострілів та мають своє власне здоров'я і якщо воно впаде до 0 – ворог зникне.



Рис. 3.6 Битва з ворогами

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

Коли противник отримує шкоду він ненадовго стає червоним (що видно на рисунку 3.6), це зроблено для наглядної демонстрації отримання шкоди ворогами.

В ході битви з ворогами, гравець також може постраждати, як мінімум це отримання шкоди (Рисунок 3.7), або ж якщо його здоров'я впаде до 0, перманентна смерть та повернення в меню прокачки персонажа.

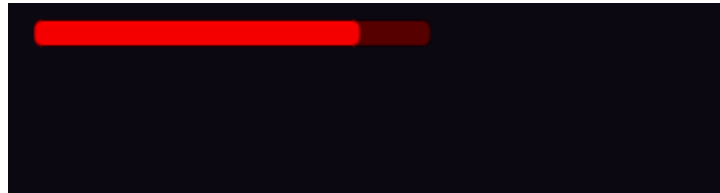


Рис. 3.6 Здоров'я гравця після отримання шкоди

Також в деяких кімнатах гравця будуть чекати скрині з нагородою, це може бути як нова зброя, так і аптечки або монети.



Рис. 3.7 Гравець поряд скринєю з нагородою

При натисканні на букву «Е», гравець відкриє цю скриню і з неї випадє випадковий предмет.

Предмети в грі діляться на 3 типи, це зброя (яка складається з 3 комплектів, поділених по своїй рідкості), аптечка та монети.

Демонстрацію як монети витглядають на землі можна побачити на рисунку 3.8.

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		47



Рис. 3.8 Гравець поряд з монетами на підлозі

Якщо гравець натисне кнопку взаємодії поряд з монетами він їх підбере та його показник валюти оновиться.



Рис. 3.9 Оновлена кількість коштів

Зброя на землі відображається наступним чином – рисунок 3.10. Вона обертається навколо своєї осі в очкуванні на те що її підберуть, або ж пройдуть повз.



Рис. 3.10 Гравець поряд зі зброєю на підлозі

Якщо гравець підбере зброю що лежить на підлозі, зброя в його руках зміниться на ту що він підібрав, а стара скинеться на підлогу (Рис 3.11).

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		48



Рис. 3.11 Підбирання нової зброї

Також зі скрині може випасти аптечка (рис. 3.12), при взаємодії з нею гравець отримає певну кількість відсутнього здоров'я.



Рис. 3.12 Аптечка зі скрині

Загалом скрині мають досить багато речей які можуть з них випасти, але завдяки випадковості, ти ніколи не знаєш який саме предмет тебе чекає, нова зброя, аптечка чи монети.



Рис. 3.13 Невелика кількість предметів доступних для випадіння

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		49

Також в грі є портал, який можна побачити на рисунку 3.14, він відповідає за перехід до наступного данжу, і так доти, доки гравець не помре або не вийде з гри, після чого, можна знову перейти до прокачки.



Рис. 3.14 Портал для переходу в наступну локацію

Після того як ми отримали трохи коштів, знову перейдемо до вікна прокачки персонажу. Тут у нас доступні певні статистичні показники для прокачки.

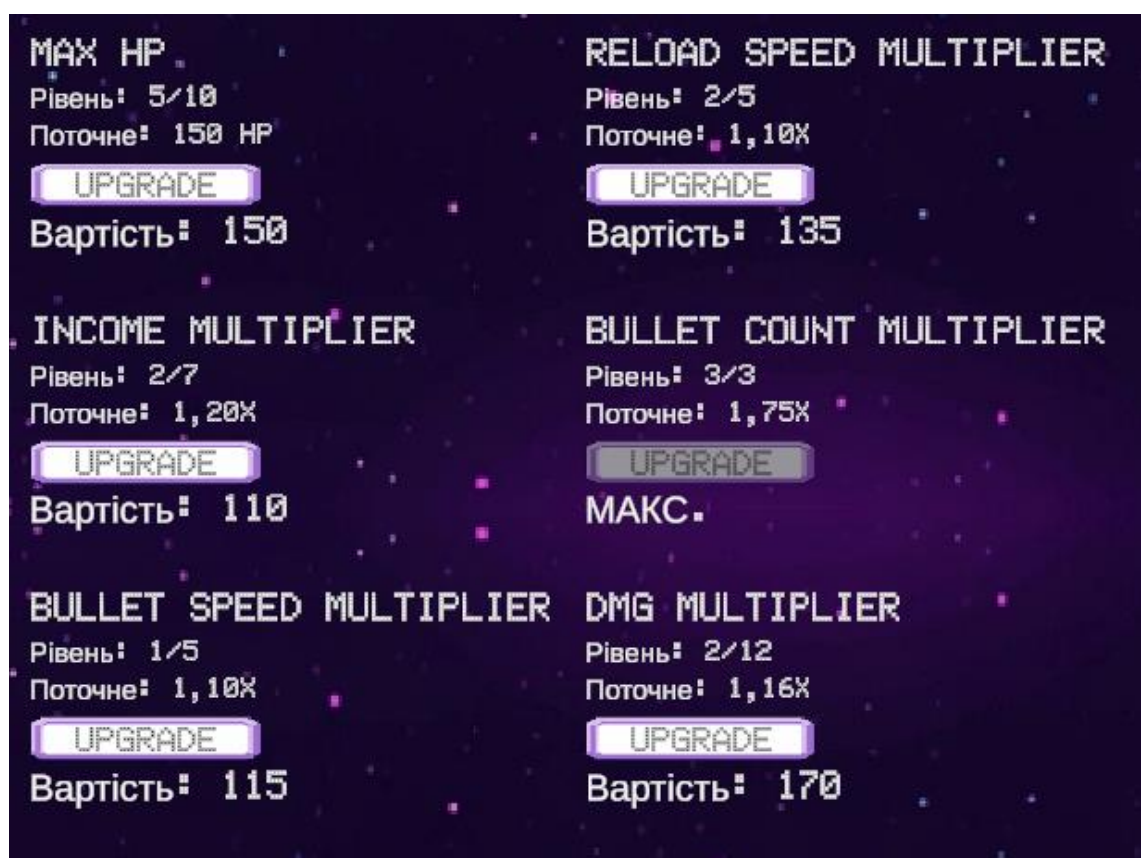


Рис. 3.15 Вікно прокачки

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				50
Змн.	Арк.	№ докум.	Підпис	Дата		

Нехай гравець хоче прокачати стат – здоров'я, після того як він натисне кнопку «Прокачати», в нього зніметься вказана кількість валюти а статистика збільшить свій рівень на 1.



Рис. 3.16 Вікно прокачки після прокачки статусу НР

3.3. Тестування програмного продукту

Тестування програмного продукту є важливою складовою процесу розробки гри, що дозволяє виявити помилки, недоліки та забезпечити стабільну роботу системи в різних ситуаціях. У межах створення гри в жанрі roguelike на Unity тестування проводилося як вручну, так і за допомогою інструментів, що доступні в самій ігровій платформі.

Основна увага приділялася перевірці працездатності ключових механік гри: руху гравця, бойової системи, стрільби, генерації рівнів, збереження й завантаження прогресу, системи покращень та взаємодії з об'єктами навколишнього середовища. Тестувалося коректне реагування гри на різні дії користувача, зокрема використання клавіш управління, стрільба, отримання шкоди, підбір предметів і смерть персонажа.

Особливо ретельно перевірявся процес генерації мап, адже саме він створює динамічне середовище, у якому гравець проводить основну частину

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		51

часу. Була проведена перевірка на коректне розміщення кімнат, ворогів, об'єктів для взаємодії та виходу з рівня.

Також здійснювалося тестування системи збереження даних, яка використовує серіалізацію ігрового прогресу. Перевірялося, чи правильно зберігаються і відновлюються основні параметри гри

У результаті тестування було виявлено і усунено ряд помилок, що дозволило забезпечити плавний ігровий процес та задовільну стабільність гри на платформі Windows.

Висновки до третього розділу

У третьому розділі було розглянуто порядок встановлення та налаштування системи гри, побудованої на рушії Unity, а також особливості взаємодії користувача з її інтерфейсом. Детально описано основні елементи управління, логіку переходів між екранами, механіку прокачки персонажа та основні аспекти ігрового процесу, включаючи стрільбу, битви, підбір предметів та генерацію локацій. Окрему увагу приділено реалізації взаємодії з ігровим середовищем та динамічному розвитку подій під час проходження рівнів. Проведене тестування підтвердило стабільну роботу гри на платформі Windows, надійність механізмів збереження даних та правильність реалізації основних ігрових функцій. Це дозволяє зробити висновок, що система є зручною для користувача, технічно стабільною та функціонально завершеною на поточному етапі розробки.

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		52

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра було спроектовано та реалізовано гру в жанрі roguelike з використанням рушія Unity. Проведено детальний аналіз особливостей жанру та вивчено популярні ігрові проекти, які мають схожу ігрову концепцію. На основі отриманих результатів були сформульовані ключові функціональні та нефункціональні вимоги до гри.

У процесі розробки створено архітектуру гри, визначено набір основних скриптів і компонентів, а також обрано технології, які найкраще підходять для реалізації генерації рівнів, системи бою, прокачки персонажа та збереження прогресу. В основі структури гри лежить поділ на незалежні модулі: менеджери рівня, даних, UI, ворогів і гравця, що забезпечує легкість підтримки та масштабування.

Особливу увагу було приділено реалізації системи збереження гри, яка дозволяє зберігати ключові параметри. Гравець має змогу поступово покращувати свого героя, взаємодіяти з випадковими подіями в кімнатах, боротися з ворогами та отримувати нагороди зі скринь.

У роботі описано логіку взаємодії користувача з інтерфейсом гри: від головного меню, сторінки прокачки до безпосередньо ігрового процесу. Зазначено поведінку ворогів, дію пострілів, принцип отримання шкоди та механіку перезарядки. Деталізовано також генерацію рівнів і використання випадковості в побудові карт.

Було проведено тестування працездатності основних функцій гри, перевірено збереження і відновлення даних, а також стабільність поведінки системи в типових сценаріях гри. Готовий продукт є повноцінним ігровим проектом, що демонструє основні принципи побудови roguelike-гри та може бути основою для подальшого розвитку або розширення функціоналу.

Розроблена гра має зрозумілий інтерфейс, динамічний ігровий процес та забезпечує стабільну роботу на платформі Windows.

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				53
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Hades [Електронний ресурс]. – Режим доступу: <https://www.supergiantgames.com/games/hades/>
2. Enter The Gungeon [Електронний ресурс]. – Режим доступу: <https://enterthegungeon.com/>
3. Dead Cells [Електронний ресурс]. – Режим доступу: <https://playdigious.com/games/dead-cells/>
4. Cellar Door Games – Rogue Legacy [Електронний ресурс]. – Режим доступу: <https://cellardoorgames.com/>
5. The Story of Rogue [Електронний ресурс]. – Режим доступу: <https://spillhistorie.no/2024/07/14/the-story-of-rogue/>
6. Unity – 2D Roguelike Game [Електронний ресурс]. – Режим доступу: <https://learn.unity.com/project/2d-roguelike-tutorial>
7. Unity Manual [Електронний ресурс]. – Режим доступу: <https://docs.unity3d.com/6000.1/Documentation/Manual/UnityManual.html>
8. Procedural Cave Generation [Електронний ресурс]. – Режим доступу: <https://learn.unity.com/project/procedural-cave-generation-tutorial>
9. Unity [Електронний ресурс]. – Режим доступу: <https://unity.com/>
10. Unity Tutorials [Електронний ресурс]. – Режим доступу: <https://learn.unity.com/tutorials>
11. Microsoft Docs — C# Guide [Електронний ресурс]. – Режим доступу: <https://learn.microsoft.com/uk-ua/dotnet/csharp/>
12. Microsoft Docs — C# Programming Guide [Електронний ресурс]. – Режим доступу: <https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/>
13. Microsoft Docs — .NET Documentation [Електронний ресурс]. – Режим доступу: <https://learn.microsoft.com/en-us/dotnet/>
14. Gamasutra / Game Developer [Електронний ресурс]. – Режим доступу: <https://www.gamedeveloper.com/>

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		54

- 15.C# in Depth [Електронний ресурс]. – Режим доступу:
<https://csharpindepth.com/>
- 16.Martin, R. Clean Code: A Handbook of Agile Software Craftsmanship. – Boston: Prentice Hall, 2008. – 464 p.
- 17.Fowler, M. Refactoring: Improving the Design of Existing Code. – 2nd ed. – Boston: Addison-Wesley, 2018. – 448 p.
- 18.Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. – Boston: Addison-Wesley, 1994. – 395 p.
- 19.Hunt, A., Thomas, D. The Pragmatic Programmer: Your Journey to Mastery. – 2nd ed. – Boston: Addison-Wesley, 2019. – 352 p.
- 20.Scocca, C. Unity 2D Game Development Cookbook. – Birmingham: Packt Publishing, 2015. – 239 p.
- 21.Hocking, J. Unity in Action: Multiplatform Game Development in C# with Unity 5. – New York: Manning Publications, 2015. – 300 p.
- 22.Schell, J. The Art of Game Design: A Book of Lenses. – 3rd ed. – Boca Raton: CRC Press, 2019. – 600 p.
- 23.Bartle, R. Designing Virtual Worlds. – Indianapolis: New Riders, 2003. – 741 p.
- 24.Nystrom, R. Game Programming Patterns. – Genever Benning, 2014. – 354 p.
- 25.Gibson Bond, J. Introduction to Game Design, Prototyping, and Development. – Boston: Addison-Wesley, 2017. – 944 p.
- 26.Richter, J. CLR via C#. – 4th ed. – Redmond: Microsoft Press, 2012. – 826 p.
- 27.Shiffman, D. The Nature of Code. – Self-published, 2012. – 498 p.
- 28.Short, T. X., Adams, T. Procedural Generation in Game Design. – Boca Raton: CRC Press, 2017. – 288 p.
- 29.Rogers, S. Level Up! The Guide to Great Video Game Design. – 2nd ed. – Hoboken: Wiley, 2014. – 552 p.
- 30.Swink, S. Game Feel: A Game Designer's Guide to Virtual Sensation. – Boca Raton: CRC Press, 2008. – 376 p.

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		55

31. Adams, E., Rollings, A. Fundamentals of Game Design. – 3rd ed. – New York: New Riders, 2014. – 624 p.

		Янушевич Д. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Власенко О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		56

Додатки

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;
using NavMeshPlus.Components;
#if UNITY_EDITOR
using UnityEditor;
#endif

public class CorridorFirstDungeonGenerator : SimpleRandomWalkDungeonGenerator
{
    [SerializeField]
    private int corridorLength = 14, corridorCount = 5;
    [SerializeField]
    [Range(0.1f, 1)]
    private float roomPercent = 0.8f;

    private HashSet<Vector2Int> floorPositions, corridorPositions;
    private Dictionary<Vector2Int, HashSet<Vector2Int>> roomDictionary = new
Dictionary<Vector2Int, HashSet<Vector2Int>>();

    [SerializeField]
    private NavMeshSurface navigationSurface;

    private void Awake()
    {
        StartCoroutine(GenerateDungeonRoutine());
    }

    public IEnumerator GenerateDungeonRoutine()
    {
        RunProceduralGeneration();
        roomVisual.PlaceItems(roomDictionary, corridorPositions);

        yield return null;

        if (navigationSurface != null)
        {
            navigationSurface.BuildNavMesh();
            Debug.Log("NavMesh built successfully!");
        }
        else
        {
            Debug.LogError("NavMeshSurface is not assigned in the Inspector for
DungeonGenerator!");
        }
    }
}

```

```

protected override void RunProceduralGeneration()
{
    tilemapVisualizer.Clear();

    if (floorPositions == null) floorPositions = new HashSet<Vector2Int>();
    if (corridorPositions == null) corridorPositions = new
HashSet<Vector2Int>();
    if (roomDictionary == null) roomDictionary = new Dictionary<Vector2Int,
HashSet<Vector2Int>>();
    floorPositions.Clear();
    corridorPositions.Clear();
    roomDictionary.Clear();

    CorridorFirstGeneration();
}

private void CorridorFirstGeneration()
{
    HashSet<Vector2Int> currentFloorPositions = new HashSet<Vector2Int>();
    HashSet<Vector2Int> currentPotentialRoomPositions = new
HashSet<Vector2Int>();

    CreateCorridors(currentFloorPositions, currentPotentialRoomPositions);

    HashSet<Vector2Int> roomPositions =
CreateRooms(currentPotentialRoomPositions);

    List<Vector2Int> deadEnds = FindAllDeadEnds(currentFloorPositions);

    CreateRoomsAtDeadEnd(deadEnds, roomPositions);

    currentFloorPositions.UnionWith(roomPositions);

    floorPositions.UnionWith(currentFloorPositions);

    tilemapVisualizer.PaintFloorTiles(floorPositions);
    WallGenerator.CreateWalls(floorPositions, tilemapVisualizer);
}

private void CreateRoomsAtDeadEnd(List<Vector2Int> deadEnds,
HashSet<Vector2Int> roomFloors)
{
    foreach (var position in deadEnds)
    {
        if (roomFloors.Contains(position) == false)
        {
            var room = RunRandomWalk(randomWalkParameters, position);
            SaveRoomData(position, room);
            roomFloors.UnionWith(room);
        }
    }
}

```

```

    }
}

private List<Vector2Int> FindAllDeadEnds(HashSet<Vector2Int>
floorPositionsToCheck)
{
    List<Vector2Int> deadEnds = new List<Vector2Int>();
    foreach (var position in floorPositionsToCheck)
    {
        int neighboursCount = 0;
        foreach (var direction in Direction2D.cardinalDirectionsList)
        {
            if (floorPositionsToCheck.Contains(position + direction))
                neighboursCount++;
        }
        if (neighboursCount == 1)
            deadEnds.Add(position);
    }
    return deadEnds;
}

private void SaveRoomData(Vector2Int roomPosition, HashSet<Vector2Int>
roomFloor)
{
    roomDictionary[roomPosition] = roomFloor;
}

private HashSet<Vector2Int> CreateRooms(HashSet<Vector2Int>
potentialRoomPositions)
{
    HashSet<Vector2Int> roomPositions = new HashSet<Vector2Int>();
    int roomToCreateCount = Mathf.RoundToInt(potentialRoomPositions.Count *
roomPercent);

    List<Vector2Int> roomsToCreate = potentialRoomPositions.OrderBy(x =>
Guid.NewGuid()).Take(roomToCreateCount).ToList();

    foreach (var roomPosition in roomsToCreate)
    {
        var roomFloor = RunRandomWalk(randomWalkParameters, roomPosition);
        SaveRoomData(roomPosition, roomFloor);
        roomPositions.UnionWith(roomFloor);
    }
    return roomPositions;
}

private void CreateCorridors(HashSet<Vector2Int> currentFloorPositions,
HashSet<Vector2Int> currentPotentialRoomPositions)
{

```

```

        var currentPosition = startPosition;
        currentPotentialRoomPositions.Add(currentPosition);

        for (int i = 0; i < corridorCount; i++)
        {
            var corridor =
ProceduralGenerationAlgorithms.RandomWalkCorridor(currentPosition, corridorLength);
            currentPosition = corridor[corridor.Count - 1];
            currentPotentialRoomPositions.Add(currentPosition);
            currentFloorPositions.UnionWith(corridor);
        }
        corridorPositions = new HashSet<Vector2Int>(currentFloorPositions);
    }
}

```

Додаток В

```

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

public static class WallTypesHelper
{
    public static HashSet<int> wallTop = new HashSet<int>
    {
        0b1111,
        0b0110,
        0b0011,
        0b0010,
        0b1010,
        0b1100,
        0b1110,
        0b1011,
        0b0111
    };

    public static HashSet<int> wallSideLeft = new HashSet<int>
    {
        0b0100
    };

    public static HashSet<int> wallSideRight = new HashSet<int>
    {
        0b0001
    };

    public static HashSet<int> wallBottom = new HashSet<int>
    {
        0b1000
    };
}

```

```
};
```

```
public static HashSet<int> wallInnerCornerDownLeft = new HashSet<int>
{
    0b11110001,
    0b11100000,
    0b11110000,
    0b11100001,
    0b10100000,
    0b01010001,
    0b11010001,
    0b01100001,
    0b11010000,
    0b01110001,
    0b00010001,
    0b10110001,
    0b10100001,
    0b10010000,
    0b00110001,
    0b10110000,
    0b00100001,
    0b10010001
};
```

```
public static HashSet<int> wallInnerCornerDownRight = new HashSet<int>
{
    0b11000111,
    0b11000011,
    0b10000011,
    0b10000111,
    0b10000010,
    0b01000101,
    0b11000101,
    0b01000011,
    0b10000101,
    0b01000111,
    0b01000100,
    0b11000110,
    0b11000010,
    0b10000100,
    0b01000110,
    0b10000110,
    0b11000100,
    0b01000010
};
```

```
public static HashSet<int> wallDiagonalCornerDownLeft = new HashSet<int>
{
    0b01000000
};
```

```

};

public static HashSet<int> wallDiagonalCornerDownRight = new HashSet<int>
{
    0b00000001
};

public static HashSet<int> wallDiagonalCornerUpLeft = new HashSet<int>
{
    0b00010000,
    0b01010000,
};

public static HashSet<int> wallDiagonalCornerUpRight = new HashSet<int>
{
    0b00000100,
    0b00000101
};

public static HashSet<int> wallFull = new HashSet<int>
{
    0b1101,
    0b0101,
    0b1101,
    0b1001
};

public static HashSet<int> wallFullEightDirections = new HashSet<int>
{
    0b00010100,
    0b11100100,
    0b10010011,
    0b01110100,
    0b00010111,
    0b00010110,
    0b00110100,
    0b00010101,
    0b01010100,
    0b00010010,
    0b00100100,
    0b00010011,
    0b01100100,
    0b10010111,
    0b11110100,
    0b10010110,
    0b10110100,
    0b11100101,
    0b11010011,
    0b11110101,

```

```

        0b11010111,
        0b11010111,
        0b11110101,
        0b01110101,
        0b01010111,
        0b01100101,
        0b01010011,
        0b01010010,
        0b00100101,
        0b00110101,
        0b01010110,
        0b11010101,
        0b11010100,
        0b10010101

    };

    public static HashSet<int> wallBottmEightDirections = new HashSet<int>
    {
        0b01000001
    };
}

```

Додаток С

RoomVisual.

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;
using UnityEngine.Experimental.GlobalIllumination;
using UnityEngine.UIElements;

public class RoomVisual : MonoBehaviour
{
    private List<GameObject> _objects = new List<GameObject>();

    private int? spawnRoomId = null;
    private int? exitRoomId = null;

    [SerializeField]
    SpawnRoom spawnRoom;

    [SerializeField]
    ExitRoom exitRoom;
}

```

```

[SerializeField]
List<EnemyRoom> enemyRooms;

public void PlaceItems(Dictionary<Vector2Int, HashSet<Vector2Int>>
roomDictionary, HashSet<Vector2Int> corridors)
{
    SetRooms(roomDictionary.Count);
    Clear();

    SpawnPlayerRoom(roomDictionary, corridors);
    SpawnExitRoom(roomDictionary, corridors);
    SpawnEnemyRooms(roomDictionary, corridors);
}

private void SpawnPlayerRoom(Dictionary<Vector2Int, HashSet<Vector2Int>>
roomDictionary, HashSet<Vector2Int> corridors)
{
    if (spawnRoomId.HasValue &&
roomDictionary.ContainsKey((Vector2Int)roomDictionary.ElementAt(spawnRoomId.Value).
Key))
    {
        var room = roomDictionary.ElementAt(spawnRoomId.Value);
        HashSet<Vector2Int> corridorsOutsideRoom =
GetRoomWOCorridors(room.Value, corridors);
        ItemPlaceHelper itemPlaceHelper = new ItemPlaceHelper(room.Value,
corridorsOutsideRoom);

        SpawnPlayer(room.Key);
        PlaceItemsToRoom(spawnRoom, itemPlaceHelper);
    }
}

private void SpawnExitRoom(Dictionary<Vector2Int, HashSet<Vector2Int>>
roomDictionary, HashSet<Vector2Int> corridors)
{
    if (exitRoomId.HasValue &&
roomDictionary.ContainsKey((Vector2Int)roomDictionary.ElementAt(exitRoomId.Value).K
ey))
    {
        var room = roomDictionary.ElementAt(exitRoomId.Value);
        HashSet<Vector2Int> corridorsOutsideRoom =
GetRoomWOCorridors(room.Value, corridors);
        ItemPlaceHelper itemPlaceHelper = new ItemPlaceHelper(room.Value,
corridorsOutsideRoom);

        PlaceItemsToRoom(exitRoom, itemPlaceHelper);
    }
}

```

```

    private void SpawnEnemyRooms(Dictionary<Vector2Int, HashSet<Vector2Int>>
roomDictionary, HashSet<Vector2Int> corridors)
    {
        int iteration = 0;
        foreach (var room in roomDictionary)
        {
            if (iteration == spawnRoomId || iteration == exitRoomId)
            {
                iteration++;
                continue;
            }

            int randomRoom = UnityEngine.Random.Range(0, enemyRooms.Count);
            HashSet<Vector2Int> corridorsOutsideRoom =
GetRoomWOCorridors(room.Value, corridors);
            ItemPlaceHelper itemPlaceHelper = new ItemPlaceHelper(room.Value,
corridorsOutsideRoom);

            SpawnEnemies(enemyRooms[randomRoom].enemyData, itemPlaceHelper);
            PlaceItemsToRoom(enemyRooms[randomRoom], itemPlaceHelper);

            iteration++;
        }
    }

    private void SpawnEnemies(List<EnemyData> enemyData, ItemPlaceHelper
itemPlaceHelper)
    {
        foreach (var item in enemyData)
        {
            int quantity = UnityEngine.Random.Range(item.MinQuantity,
item.MaxQuantity + 1);

            for (int i = 0; i < quantity; i++)
            {
                PlacementType pt = PlacementType.OpenSpace;
                if (UnityEngine.Random.Range(0, 100) < 20)
                {
                    pt = PlacementType.NearWall;
                }

                Vector2? position = itemPlaceHelper.GetItemPlacementPosition(pt,
10, new Vector2Int(1, 1), true);

                if (position != null)
                {
                    Vector2Int positionToPlace = new
Vector2Int((int)position.Value.x, (int)position.Value.y);
                    SpawnObject(item.Enemy, positionToPlace);
                }
            }
        }
    }

```

```

        }
    }

}

private void SpawnPlayer(Vector2Int spawnPoint)
{
    if (spawnRoom.player != null)
    {
        GameObject player = GameObject.FindGameObjectWithTag("Player");
        if (player == null)
        {
            SpawnObject(spawnRoom.player, spawnPoint, false);
        }
        else
        {
            player.transform.position = new Vector3(spawnPoint.x, spawnPoint.y,
0);
        }
    }
    else
    {
        Debug.Log("error");
    }
}

private void SpawnObject(GameObject objectToSpawn, Vector2Int spawnPoint, bool
addToObjList = true)
{
    Vector3 position3D = new Vector3(spawnPoint.x + 0.5f, spawnPoint.y + 0.5f,
0f);
    GameObject objectInstance = Instantiate(objectToSpawn, position3D,
Quaternion.identity);
    if (addToObjList)
    {
        _objects.Add(objectInstance);
    }
}

private void PlaceItemsToRoom(Room roomType, ItemPlaceHelper itemPlaceHelper)
{
    foreach (var item in roomType.itemData)
    {
        int quantity = UnityEngine.Random.Range(item.MinQuantity,
item.MaxQuantity + 1);

        for (int i = 0; i < quantity; i++)
        {
            var pt = PlacementType.NearWall;

```

```

        if (item.Item.size != new Vector2Int(1, 1))
        {
            pt = PlacementType.OpenSpace;
        }

        Vector2? position = itemPlaceHelper.GetItemPlacementPosition(pt,
10, item.Item.size, true);

        if (position != null)
        {
            Vector2Int positionToPlace = new
Vector2Int((int)position.Value.x, (int)position.Value.y);
            SpawnObject(item.Item.item, positionToPlace);
        }
    }
}

private static HashSet<Vector2Int> GetRoomWOCorridors(HashSet<Vector2Int> room,
HashSet<Vector2Int> corridors)
{
    HashSet<Vector2Int> corridorsOutsideRoom = new HashSet<Vector2Int>();

    foreach (var corridor in room)
    {
        if (!corridors.Contains(corridor))
        {
            corridorsOutsideRoom.Add(corridor);
        }
    }

    return corridorsOutsideRoom;
}

private void SetRooms(int count)
{
    spawnRoomId = UnityEngine.Random.Range(0, count);
    while (exitRoomId == null)
    {
        int id = UnityEngine.Random.Range(0, count);
        if (id != spawnRoomId)
        {
            exitRoomId = id;
        }
    }
}

public void Clear()
{
    foreach (var obj in _objects)

```

```

        {
            if (obj != null)
            {
                Destroy(obj);
            }
        }
        _objects.Clear();
    }
}

```

ItemPlaceHelper.

```

using System.Collections;
using System.Collections.Generic;
using System.Linq;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.UIElements;

public class ItemPlaceHelper
{
    public bool showOccupiedTiles = true;
    public Color occupiedTileColor = Color.red;

    Dictionary<PlacementType, HashSet<Vector2Int>> tileByType = new
Dictionary<PlacementType, HashSet<Vector2Int>>();

    HashSet<Vector2Int> roomFloorNoCorridor;

    public ItemPlaceHelper(HashSet<Vector2Int> roomFloor, HashSet<Vector2Int>
roomFloorNoCorridor)
    {
        Graph graph = new Graph(roomFloor);

        this.roomFloorNoCorridor = roomFloorNoCorridor;

        foreach (var position in roomFloorNoCorridor)
        {
            int neighboursCount8Dir =
graph.GetNeighbours8Direction(position).Count;
            PlacementType type = neighboursCount8Dir < 8 ? PlacementType.NearWall :
PlacementType.OpenSpace;

            if (tileByType.ContainsKey(type) == false)
            {
                tileByType[type] = new HashSet<Vector2Int>();
            }

```

```

        if (type == PlacementType.NearWall &&
graph.GetNeighbours4Direction(position).Count < 4)
        {
            continue;
        }

        tileByType[type].Add(position);
    }
}

public Vector2? GetItemPlacementPosition(PlacementType type, int iterationsMax,
Vector2Int size, bool addOffset)
{
    int itemArea = size.x * size.y;

    if (tileByType[type].Count < itemArea)
    {
        return null;
    }

    int iteration = 0;
    while (iteration < iterationsMax)
    {
        iteration++;
        int index = Random.Range(0, tileByType[type].Count);
        Vector2Int position = tileByType[type].ElementAt(index);

        if (itemArea > 1)
        {
            var (result, placementPosition) = PlaceBigItem(position, size,
addOffset);

            if (result == false)
            {
                continue;
            }

            tileByType[type].ExceptWith(placementPosition);
            tileByType[PlacementType.NearWall].ExceptWith(placementPosition);
        }
        else
        {
            tileByType[type].Remove(position);
        }

        return position;
    }
    return null;
}

```

```

        private (bool, List<Vector2Int>) PlaceBigItem(Vector2Int originalPosition,
Vector2Int size, bool addOffset)
        {
            List<Vector2Int> positions = new List<Vector2Int>();
            int maxX = addOffset ? size.x + 1 : size.x;
            int maxY = addOffset ? size.y + 1 : size.y;
            int minX = addOffset ? -1 : 0;
            int minY = addOffset ? -1 : 0;

            for (int row = minX; row <= maxX; row++)
            {
                for (int col = minY; col <= maxY; col++)
                {
                    if (col == 0 && row == 0) continue;

                    Vector2Int newPosToCheck = new Vector2Int(originalPosition.x + row,
originalPosition.y + col);

                    if (roomFloorNoCorridor.Contains(newPosToCheck) == false)
                    {
                        return (false, positions);
                    }
                    positions.Add(newPosToCheck);
                }
            }
            return (true, positions);
        }
    }

    public enum PlacementType
    {
        OpenSpace, NearWall
    }
}

```

Додаток Е

```

using System;
using System.Collections;
using System.Collections.Generic;
using TMPro;
using Unity.VisualScripting.Antlr3.Runtime.Misc;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class Player : MonoBehaviour, IDataPersistence
{
    [SerializeField] private float _moveSpeed;
}

```

```

[SerializeField] private float _interactionRadius = 1f;
[SerializeField] private string targetLayerName = "Interactable";
[SerializeField] private GameObject gunGO;

[SerializeField] private TextMeshProUGUI coins;
[SerializeField] private Slider HPBar;

private PlayerStats playerStats = new PlayerStats();
private int currentHp;

private Rigidbody2D _rigidBody;
private Vector2 mousePos;

private GameObject _closestInteractable;

private Gun _gun;

public Vector2 GetDirection()
{
    var aY = Convert.ToInt32(Input.GetKey(KeyCode.W)) -
Convert.ToInt32(Input.GetKey(KeyCode.S));
    var aX = Convert.ToInt32(Input.GetKey(KeyCode.D)) -
Convert.ToInt32(Input.GetKey(KeyCode.A));

    Vector2 dir = new Vector2(aX, aY);

    return dir.normalized;
}

private void Move(Vector2 vector)
{
    _rigidBody.velocity = vector * _moveSpeed;
}

void Start()
{
    _rigidBody = GetComponent<Rigidbody2D>();

    currentHp = playerStats.MaxHp;
}

void Update()
{
    Move(GetDirection());

    mousePos = Camera.main.ScreenToWorldPoint(Input.mousePosition);

    float angel = Mathf.Atan2(mousePos.y - transform.position.y, mousePos.x -
transform.position.x) * Mathf.Rad2Deg - 90f;

```

```

transform.localRotation = Quaternion.Euler(0, 0, angel);

if (Input.GetMouseButton(0))
{
    _gun.Shoot();
}

if (Input.GetKeyDown(KeyCode.Q))
{
    TakeDmg(10);
}

FindClosestObjectInLayer(targetLayerName, _interactionRadius);
if (Input.GetKeyDown(KeyCode.E))
{
    InteractWithClosest();
}
}

private void FindClosestObjectInLayer(string targetLayerName, float radius)
{
    int targetLayer = LayerMask.NameToLayer(targetLayerName);
    Collider2D[] hits = Physics2D.OverlapCircleAll(transform.position, radius);

    Collider2D closest = null;
    float closestDistance = Mathf.Infinity;

    foreach (var hit in hits)
    {
        if (hit.gameObject.layer == targetLayer)
        {
            float distance = Vector2.Distance(transform.position,
hit.transform.position);
            if (distance < closestDistance)
            {
                closestDistance = distance;
                closest = hit;
            }
        }
    }

    _closestInteractable = closest != null ? closest.gameObject : null;
}

private void InteractWithClosest()
{
    if (_closestInteractable == null) return;

    Interactable interactable =
_closestInteractable.GetComponent<Interactable>();

```

```

        if (interactable != null)
        {
            interactable.Interact();
        }
    }

    private void SetHPBarSetting()
    {
        HPBar.maxValue = playerStats.MaxHp;
        HPBar.value = HPBar.maxValue;
    }

    private void UpdateHpBar()
    {
        HPBar.value = currentHp;
    }

    private void Die()
    {
        DataPersistenceManager manager =
FindObjectOfType<DataPersistenceManager>();
        if (manager != null)
        {
            manager.SaveGame();
        }
        SceneManager.LoadScene("LvlUpPage");
    }

    private void UpdateCoins()
    {
        coins.text = playerStats.coins.ToString();
    }

    public void AddCoins(int coins)
    {
        float multiplier = playerStats?.IncomeMultiplier ?? 1f;
        int totalCoins = Mathf.RoundToInt(coins * multiplier);

        playerStats.coins += totalCoins;
        UpdateCoins();
    }

    public void TakeDmg(int dmg)
    {
        currentHp -= dmg;
        UpdateHpBar();
        if (currentHp <= 0)
        {
            Die();
        }
    }

```

```

    }
}

public void Heal(int hp)
{
    var newHp = currentHp + hp;
    if(newHp >= playerStats.MaxHp)
    {
        currentHp = playerStats.MaxHp;
    }
    else
    {
        currentHp = newHp;
    }
    UpdateHpBar();
}

public void LoadData(GameData data)
{
    playerStats = data.Stats;
    UpdateCoins();
    currentHp = playerStats.MaxHp;
    SetHPBarSetting();
    UpdateHpBar();
    _gun = gunGO.GetComponent<Gun>();
    _gun.Initialize(playerStats);
}

public void SaveData(ref GameData data)
{
    data.Stats = playerStats;
}
}

```