**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**
**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ**
**ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**
**КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

# КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка гри Card risk в жанрі Roguelike засобами рушія Godot мовою GDScript»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

*Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело*

_____ Анжеліка КУРІС
*(підпис)*

Виконала: здобувачка вищої освіти групи ПД-42

Анжеліка КУРІС

Керівник: Оксана ЗОЛОТУХІНА
*к.т.н., доцент*

Рецензент: _____

**Київ 2024**

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

**Навчально-науковий інститут інформаційних технологій**

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

**ЗАТВЕРДЖУЮ**
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

# ЗАВДАННЯ
# НА КВАЛІФІКАЦІЙНУ РОБОТУ

Куріс Анжеліці Сергіївні

1. Тема кваліфікаційної роботи: «Розробка гри Card risk в жанрі Roguelike засобами рушія Godot мовою GDScript»
керівник кваліфікаційної роботи к.т.н., доцент Оксана ЗОЛОТУХІНА,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: документація Godot, документація GDScript

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області для створення гри "Card Risk" в жанрі “Roguelike”.
2. Проектування гри "Card Risk" в жанрі “Roguelike.
3. Програмна реалізація гри “Card Risk”.
4. Тестування застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Концепт гри.
3. Вимоги до програмного забезпечення.
4. Програмні засоби та інструменти реалізації.
5. Діаграма варіантів виконання.
6. Діаграма діяльності ігрового процесу.
7. Діаграма класів сутностей.
8. Діаграма класів рівня та боса.
9. Екранні форми.
10. Демонстрація роботи застосунку.
11. Апробація результатів дослідження.

6. Дата видачі завдання «28» лютого 2024 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Підбір та аналіз науково-технічної літератури | 28.02.-06.03.2024 | |
| 2 | Аналіз та дослідження існуючих аналогів | 07.03-13.03.2024 | |
| 3 | Огляд та аналіз засобів реалізації | 14.03-17.03.2024 | |
| 4 | Проектування гри “Card risk” | 18.03-21.03.2024 | |
| 5 | Програмна реалізація гри “Card risk” | 22.03-22.04.2024 | |
| 6 | Тестування гри “Card risk” | 23.04-28.04.2024 | |
| 7 | Оформлення роботи: вступ, висновки, реферат | 29.04-05.05.2024 | |
| 8 | Розробка демонстраційних матеріалів | 06.05-12.05.2024 | |
| 9 | Попередній захист роботи | 13.05-31.05.2024 | |

Здобувачка вищої освіти _____ Анжеліка КУРІС
*(підпис)*

Керівник
кваліфікаційної роботи _____ Оксана ЗОЛОТУХІНА
*(підпис)*

# РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 40 стор., 1 табл., 23 рис., 5 джерел.

*Мета роботи* – покращення геймплею в жанрі “Roguelike” з використанням ігрового рушія Godot та мови Gdscript.

*Об’єкт дослідження* – геймплей в жанрі “Roguelike”.

*Предмет дослідження* – гра в жанрі “Roguelike”.

*Короткий зміст роботи*: В роботі проведено аналіз існуючих ігор в жанрі "Roguelike", визначено концепт гри, особливості геймплею та вимоги до розробки гри. Для покращення геймплею концепт гри включає появу випадкових карт для посилення сил персонажу. Розроблено алгоритми, що забезпечують реалізацію вимог, в тому числі, алгоритми для перманентної смерті та процедурного генерування рівнів. При розробці гри використано програмне забезпечення “Aseprite” для створення основних 2D об`єктів гри, основні ігрові механіки реалізовані за допомогою ігрового рушія Godot та мови програмування GDScript. Розроблено та перевірено виконання тестових сценаріїв для тестування гри «Card risk», які включають кейси тестування інтерактивності та ігрового процесу, тестування графічних компонентів та звукового супроводу

Сферою використання застосунку є розваги та розвиток гравців у жанрі Roguelike.

КЛЮЧОВІ СЛОВА: ГРА, ROGUELIKE, ГЕЙМПЛЕЙ, ПРОЦЕДУРНЕ ГЕНЕРУВАННЯ, ВИПАДКОВІСТЬ, КАРТИ.

**ЗМІСТ**

# ВСТУП

У сучасному світі ігри сприймаються як важливий елемент культури й розваг. Завдяки стрімкому розвитку технологій і підвищенню доступності, ігрова індустрія стала масовим явищем, що об'єднує мільярди людей по всьому світу. Сьогодні ігри не лише цікаві, але й дають можливість спілкуватися, проявляти творчі здібності й навіть вчитися. Розробка ігор - це галузь, що постійно еволюціонує, відкриваючи нові можливості для творчості та інновацій. Графіка, інтерактивність та доступність для гравців постійно покращуються в ігровій індустрії, що свідчить про її постійний розвиток. Це відображається в зростаючій популярності і важливості цього сегменту ринку.

Об’єкт дослідження – геймплей в жанрі “Roguelike”.

Предмет дослідження – гра в жанрі “Roguelike”.

Мета роботи – покращення геймплею в жанрі “Roguelike” використанням ігрового рушія Godot та мови Gdscript.

Завдання дослідження:

1. Провести аналіз та дослідження існуючих ігор в жанрі “Roguelike”.

2. Визначити концепт гри, особливості геймплею та вимоги до розробки гри в жанрі “Roguelike”.

3. Розробити інтерфейс та візуальне оформлення гри “Card risk”.

4. Розробити та програмно реалізувати ігрові механіки гри в жанрі “Roguelike” за допомогою рушія Godot та мови Gdscript.

5. Провести тестування гри “Card risk”.

Практична значущість дослідження полягає у покращенні ігрового досвіду гравців – початківців.

Завдяки простому геймплею, яскравій графіці та впровадженню механіки карт, які дають різні ефекти, гравці-початківці зможуть покращити свої навички в проходженні ігор у жанрі "Roguelike".

Галузь використання – розваги та розвиток гравців у жанрі Roguelike.

Робота пройшла апробацію на IV Всеукраїнській науково-практичній конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті»(15.05.2024, ДУІКТ, м. Київ) та V Всеукраїнській науково-практичній конференції молодих вчених та здобувачів вищої освіти присвяченої Дню науки (17 травня 2024 р. м. Херсон-Кропивницький). За результатами участі опубліковано тези доповідей [4, 5].

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Історія та розвиток перших відеоігор

Однією з найперших відеоігор є “Bertie the Brain”, яку розробив Джозеф Кейтс у 1950 році. Ця гра, що мала вражаючий розмір у 13 футів у висоту, була призначена для гри у хрестики-нулики. Вона була представлена на Національній виставці Канади, де привертала увагу численних відвідувачів своєю новаторською природою та великими розмірами.

“Bertie the Brain” мала кілька рівнів складності, що дозволяло відвідувачам познайомитися зі штучним інтелектом гри на різних рівнях. Це робило гру схожою на сучасні відеоігри, де рівні складності дозволяють гравцям поступово вдосконалювати свої навички.

Незважаючи на свій інноваційний характер, “Bertie the Brain” була демонтована в кінці виставки і сприймалася скоріше як новинка, ніж як серйозне технічне досягнення. На жаль, це призвело до того, що ця важлива частина історії відеоігор була значною мірою забута.

"Bertie the Brain” був важливою частиною історії гри, але він не повністю відповідав точному визначенню того, що таке відеоігра. Цікаво, що його механізм багато в чому можна порівняти з багатьма казуальними іграми нашого часу. На рисунку 1.1 зображено як Комік Денні Кей переміг “Bertie the Brain”.


Рис.1.1 Комік Денні Кей щойно переміг “Bertie the Brain”

Першою відеогрою вважається "Tennis for Two", розроблена Вільямом Гігінботамом у 1958 році. "Tennis for Two" стала першою грою, яка повністю відповідала визначенню відеогри, оскільки вона мала інтерактивний елемент та візуальний відгук на дії гравців. Це проривне досягнення стало важливою віхою в історії ігрової індустрії і проклало шлях для подальшого розвитку відеоігор.

"Tennis for Two" була продемонстрована під час дня відкритих дверей у Брукхейвенській національній лабораторії у жовтні 1958 року. Використовувався осцилограф для візуалізації тенісного корту у вигляді збоку, що дозволяло гравцям керувати грою за допомогою контролера. У кожного гравця була кнопка і ручка, за допомогою яких він міг подавати м'яч і відповідним чином змінювати кут удару.

Гра мала дуже простий, але ефективний інтерфейс, де м'яч рухався по вигнутій траєкторії, відбиваючись від поверхні корту. Незважаючи на свою простоту, "Tennis for Two" стала справжньою сенсацією дня відкритих дверей, привертаючи величезну кількість відвідувачів, які стояли в черзі, щоб зіграти. Навіть сам Гігінботам був здивований успіхом своєї розробки. Як виглядала гра "Tennis for Two" можна побачити на рисунку 1.2.

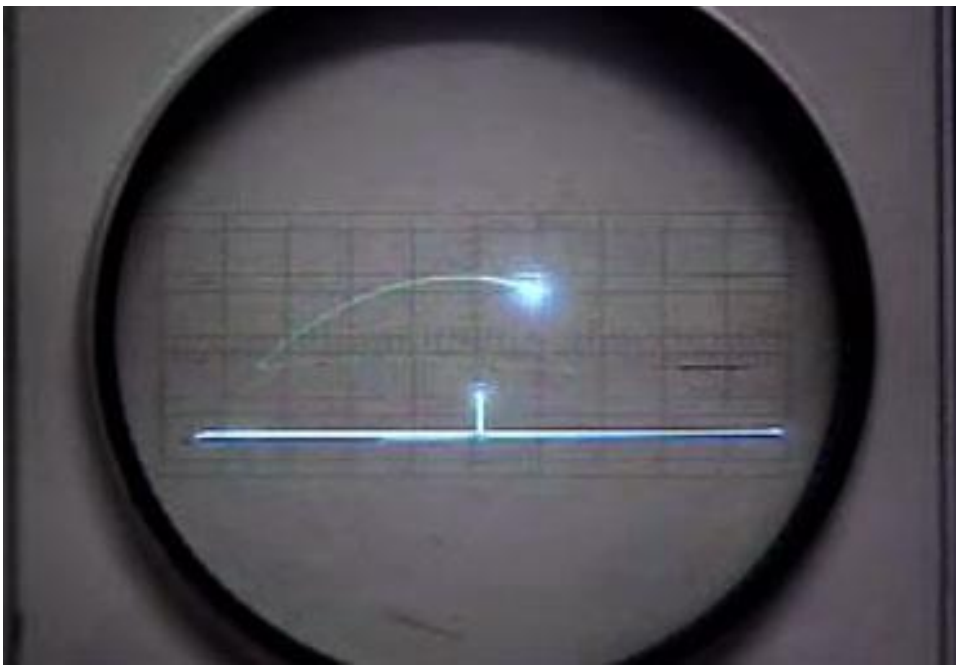


Рис.1.2 "Tennis for Two" — перша відеогра

### 1.2 Вплив сучасних технологій на ігрову індустрію

З розвитком технологій в останні десятиліття ігрова індустрія зазнала значних змін. Впровадження новітніх технологій в таких областях, як візуальні ефекти, штучний інтелект і віртуальна реальність, створило нові можливості для розробників і гравців.

Одним з ключових напрямків технологічного прогресу в ігровій індустрії є поліпшення графіки і візуальних ефектів. Сучасний графічний движок дозволяє створювати вражаючі світи з реалістичною графікою, динамічним освітленням і реалістичною фізикою. Це збільшує занурення гравця в ігровий світ, роблячи геймплей більш імерсивним і захоплюючим.

Впровадження штучного інтелекту в ігрову індустрію також має помітний вплив. Розробники використовують штучний інтелект для створення розумних і непередбачуваних супротивників, реалістичної поведінки персонажів і розробки складних інтерактивних систем. Це робить ігровий процес більш цікавим і захоплюючим і спонукає гравців розробляти стратегії і тактику для досягнення успіху в грі.

Крім того, віртуальна реальність відкриває нові горизонти для ігрової індустрії, дозволяючи гравцям зануритися в ігровий світ з повним зануренням. Це створює унікальну можливість для отримання ігрового досвіду, коли гравці можуть взаємодіяти з навколишнім середовищем і персонажами так, як будь-то вони знаходилися в самому центрі гри.

В результаті впровадження сучасних технологій в ігрову індустрію відкриває безліч нових можливостей для створення захоплюючих ігрових вражень для гравців.

### 1.3 Загальний огляд жанру “Roguelike” та його особливості

Жанр " Roguelike "зародився в 1980-х роках і отримав свою назву від гри" Rogue", яка поклала початок цьому жанру. Rogue був заснований у 1980 році Майклом Тейлором та Гленном Вікером і відомий своїми випадково створеними

картами, постійною смертю та високою складністю. Ці елементи стали визначальними для жанру "Roguelike" і перейшли в багато інших ігор цієї категорії. Знімок екрану з гри “Rogue” зображено на рисунку 1.3.

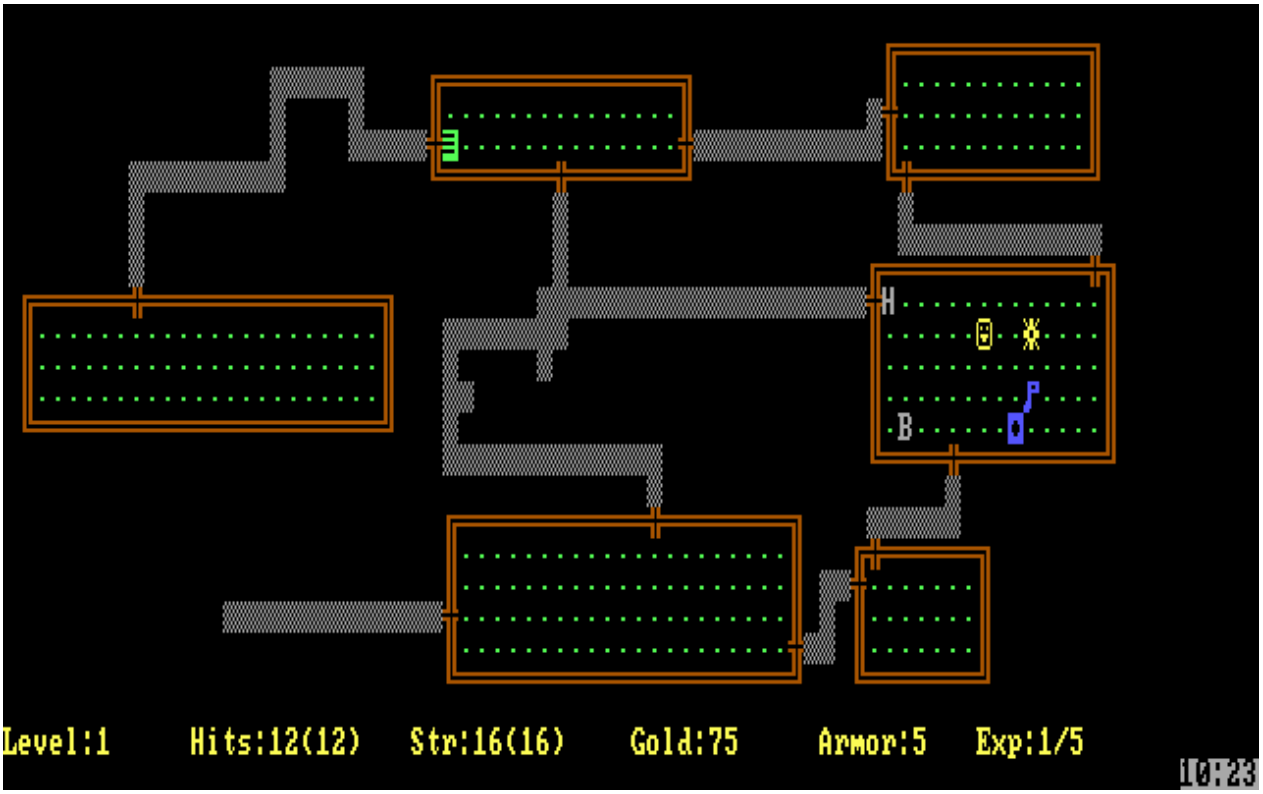



Рис.1.3 Знімок екрану із гри “Rogue”

Однією з головних особливостей Roguelike є випадково генеровані рівні. Це означає, що в кожній новій грі є унікальна карта, яка створює непередбачуваність і вимагає від гравця кожного разу адаптуватися до нових умов.

Ще однією особливістю "Roguelike" є перманентна смерть гравця. Це означає, що коли персонаж помирає, гравець втрачає весь прогрес і повертається до початку гри. Це створює чудовий стимул для гравців бути обережними та стратегічними у своїх діях.

Крім того," Roguelike " часто відрізняється високим рівнем складності, багаторівневою системою прокачування персонажів і великою кількістю ворогів і предметів. Всі ці елементи додають глибину і різноманітність ігрового процесу.

Вцілому, акцент "Roguelike" на стратегії, випадковості та високій складності робить цей жанр ідеальним вибором для любителів складних інтелектуальних

завдань у світі відеоігор. Ці елементи створюють унікальний ігровий досвід, де кожне рішення має велике значення, а випадковість гарантує, що навіть досвідчені гравці завжди зустрінуть нові виклики. Крім того, висока складність стимулює розвиток стратегічного мислення та пошук оптимальних рішень в умовах обмеженого часу та ресурсів. Такий підхід до геймплею приваблює тих, хто шукає не лише розвагу, але й виклик для свого розуму, роблячи "Roguelike" одним з найцікавіших та найбільш захоплюючих жанрів в світі відеоігор.

### 1.4 Вплив жанру “Roguelike” на ігрову індустрію

Жанр, подібний до "Roguelike", мав величезний вплив на ігрову індустрію завдяки унікальному поєднанню випадкових рівнів, постійної смерті та високої складності. З успіхом "Roguelike" інші розробники почали створювати власну інтерпретацію цього жанру, що розширило межі ігрового дизайну. Концепція постійної смерті стала нормою в багатьох іграх, заохочуючи гравців бути обережними та мислити стратегічно. "Roguelike" також застосував новий підхід до ігрового процесу, який робить кожну гру унікальною завдяки випадково створеним рівням. Розвиток жанру триває, з'являються нові ідеї і механізми, які постійно цікавлять гравців в цьому жанрі. В цілому, "Roguelike" збагатив ігрову індустрію різноманітністю та інноваціями, створюючи захоплюючі ігрові можливості для гравців по всьому світу.

### 1.5 Жанрове змішування з “Roguelike” іграми

Жанр "Roguelike" має значний вплив на ігрову індустрію, створюючи риси інновацій та інтересу серед гравців. Одним із найцікавіших аспектів його впливу є його змішування з іншими жанрами, що створює можливість для створення нових і захоплюючих ігор. Ось кілька способів, якими "Roguelike" поєднується з іншими жанрами, щоб підвищити інтерес гравців і розширити межі ігрового досвіду.

— Поєднання з RPG: Гравці, які люблять глибокий сюжет та розвиток персонажів, можуть занурюватися в гру завдяки введенню елементів RPG,

таких як розвиток персонажів, складні сюжети та діалогові системи. Прикладом є "Hades", яка поєднує швидкий та випадковий геймплей "Roguelike" з глибоким сюжетом, багатими персонажами та їх розвитком. Скріншот гемплею гри "Hades" можна побачити на рисунку 1.4.


Рис.1.4 Геймплей гри “Hades”

— Поєднання з картковими іграми: Інтеграція механік побудови колоди та стратегічного планування залучає гравців, які цінують тактичне мислення та довгострокове планування. "Slay the Spire" комбінує стратегічну карткову механіку з "Roguelike" елементами, де кожен забіг унікальний завдяки новим картам та комбінаціям. Скріншот гемплею гри "Slay the Spire" можна побачити на рисунку 1.5.


Рис.1.5 Геймплей гри "Slay the Spire"

— Поєднання з фермерськими симуляторами: Гравці, які люблять управління ресурсами та побудову, можуть насолоджуватися грою завдяки елементам фермерського симулятора. "Cult of the Lamb" поєднує елементи "Roguelike" з механіками фермерського симулятора, де гравці керують власним культом, вирощують ресурси та досліджують небезпечні підземелля, створюючи унікальний та захоплюючий ігровий досвід. Скріншот гемплею гри "Cult of the Lamb" можна побачити на рисунку 1.6.

— Рис.1.6 Геймплей гри "Cult of the Lamb"

Змішування жанру "Roguelike" з іншими жанрами дозволяє створити більш різноманітний ігровий досвід, тим самим розширюючи цільову аудиторію.

### 1.6 Аналіз існуючих ігор в жанрі “Roguelike” на ринку

Жанр "Roguelike" продовжує залишатися популярним серед гравців завдяки своїй високій іграбельності, складної ігрової механіки і захоплюючому сюжету. Ці ігри відомі своїми випадково генерованими рівнями, постійною загибеллю персонажів і можливістю отримувати нові ігрові враження при кожному проходженні. Давайте розглянемо кілька відомих представників цього жанру на сучасному ринку.

"The Binding of Isaac: Rebirth" - динамічна рогаликовая гра з елементами стрільби, розроблена відомими незалежними розробниками Едмундом Макмілленом і Nicalis Studio. Гра була випущена в 2014 році і успішно перезапустила оригінальну гру "The Binding of Isaac", яка була випущена в 2011 році. Геймплей гри показано на рисунку 1.4.


Рис.1.4 Геймплей гри "The Binding of Isaac: Rebirth"

Головний герой гри-Хлопчик на ім'я Айзек, який тікає від своєї матері в підземеллі, де йому належить зіткнутися з різними монстрами, ворогами і босами. Ця підземна пригода пов'язана з темами релігії, жертвопринесення та віри, що робить гру дещо загадковою та трохи зловісною.

У грі гравцеві належить пройти через підземелля, збираючи предмети, які можуть поліпшити його здоров'я, атаку, захист або інші характеристики. Кожен рівень генерується випадковим чином, і кожна гра буде унікальною. Боротьба з босами та звичайними ворогами вимагає від гравця стратегії та майстерності.

Однією з переваг "The Binding of Isaac: Rebirth" є випадковій генерації рівнів. Кожна гра унікальна завдяки випадковому генеруванню рівнів, ворогів і предметів, що забезпечує безперервний інтерес і непередбачуваність в кожному проходженні. Крім того, гра має глибокий геймплей з багатьма різними предметами, що взаємодіють між собою, створюючи унікальні комбінації та стратегії.

Гра відзначається високою реіграбельністю. Завдяки випадковому генеруванню та численним предметам, гравці можуть повертатися до гри знову і знову, кожного разу отримуючи новий досвід. Захоплюючий сюжет, пов'язаний з темами релігії, жертвопринесення та віри, додає грі глибини та інтересу, роблячи її не просто розвагою, а й викликом для розуму.

Атмосферна графіка та музика створюють неповторну атмосферу, що додає позитивне враження до загального ігрового досвіду. Таким чином, "The Binding of Isaac: Rebirth" є чудовим прикладом успішної гри в жанрі "Roguelike", яка продовжує приваблювати гравців своїм глибоким ігровим процесом, високою реіграбельністю та унікальним стилем.

Ще одна відома гра в жанрі "Roguelike", яка заслуговує на особливу увагу, - це "Enter the Gungeon". Це динамічний шутер з елементами рогалика, розроблений студією Dodge Roll і випущений у 2016 році. Гра швидко завоювала популярність завдяки своїй унікальній механіці, стильній графіці та захоплюючому геймплею. Скріншот геймплея гри "Enter the Gungeon" зображено на рисунку 1.5.

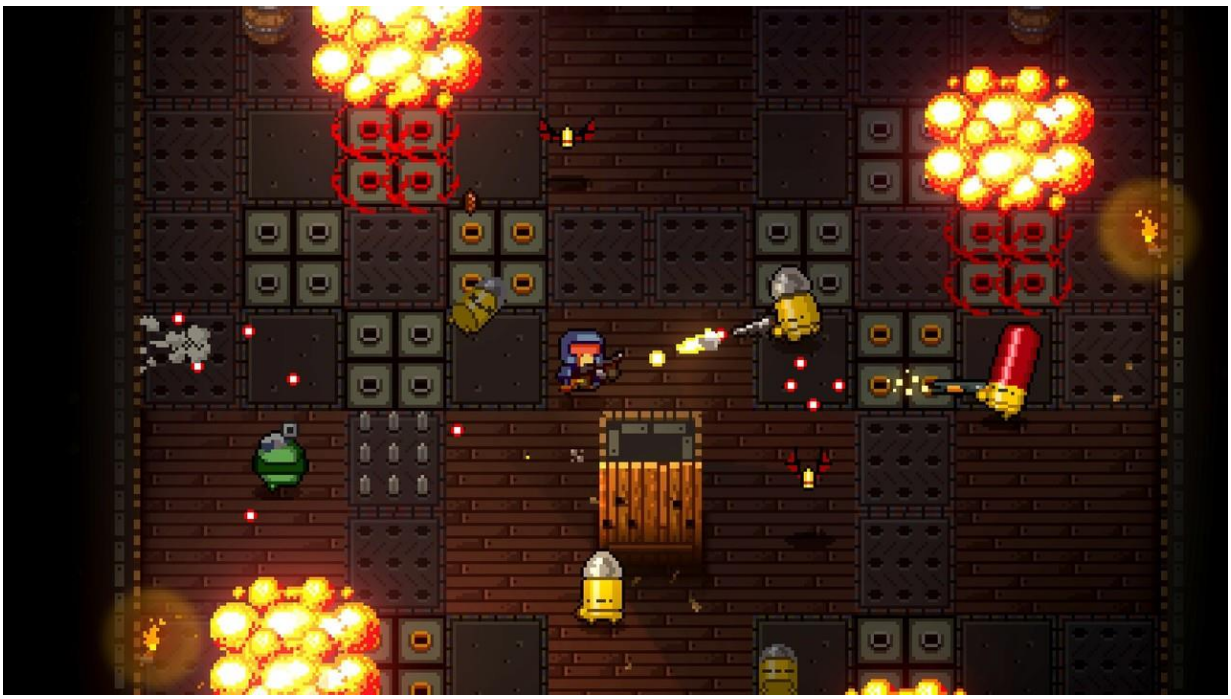


Рис.1.5. Геймплей гри "Enter the Gungeon"

У грі "Enter the Gungeon" гравці мають можливість дослідити підземелля, відоме як "Gungeon". Це місце, що складається з безлічі кімнат, коридорів і секретних зон, які генеруються випадковим чином при кожному новому проходженні. Підземелля Гангіон відоме своїми складними лабіринтами та небезпечними монстрами, що чекають на гравців на кожному кроці.

Головною метою гравців є подолання підземелля, досягнення кінцевого боса та виживання у цьому небезпечному середовищі. Під час свого подорожування гравці збирають різноманітні зброї, предмети та бонуси, які можуть допомогти їм у битві з ворогами. Також вони можуть зустріти торговців, які продають корисні предмети, або зайти до секретних кімнат, де можна знайти цінні ресурси.

Однією з головних переваг "Enter the Gungeon" є багатий вибір зброї. У грі представлено понад 300 видів зброї і предметів, кожен з яких володіє своїми унікальними характеристиками і ефектами. Це дозволяє гравцям експериментувати з різними комбінаціями зброї та предметів та створювати унікальні стратегії проходження підземелля.

У гру також дуже легко грати. Завдяки випадково генерованим рівням і різноманітності зброї, кожне нове проходження приносить нові враження і випробування. Це спонукає гравців повертатися до гри знову і знову.

"Enter the Gungeon" відрізняється своєю складністю і глибоким ігровим процесом. Гравцям потрібно не тільки влучно стріляти, але і ухилятися від численних куль і атак противника, що створює захоплюючий і динамічний ігровий процес. Ігрові боси вимагають розробки спеціальних тактик і стратегій, щоб перемогти їх.

Графіка і музичний супровід гри створюють атмосферу ретро-стилю і надають грі особливий шарм. Деталізована піксельна графіка і звуковий супровід роблять гру "Enter the Gungeon" візуально привабливою і захоплюючою на слух.

Таблиця 1.1

Порівняльна таблиця

| **Критерії порівняння** | **The Binding of Isaac** | **Enter the Gungeon** | **Card risk** |
|---|---|---|---|
| Системні вимоги | 2 GB RAM;<br>Intel Core 2 Duo;<br>Discreet video card | 2 GB RAM;<br>Intel Core 2 Duo;<br>GeForce 7600 GS | 2 GB RAM;<br>Intel Core 2 Duo;<br>OpenGL 3.3 |
| Наявність процедурної генерації рівнів | + | + | + |
| Складність геймплею | Складна | Складна | Середня |
| Тривалість ігрової сесії | 15 – 30 хвилин | 15 – 30 хвилин | До 10 хвилин |
| Елемент ризику з використанням грального карт | - | - | + |
| Інтуїтивно зрозуміле управління | - | + | + |

# 2 ОБГРУНТУВАННННЯ ЗАСОБІВ РЕАЛІЗАЦІЇ

## 2.1 Ігровий рушій Godot

Ігровий движок Godot є одним з найпопулярніших інструментів розробки відеоігор у сучасній ігровій індустрії. Він примітний не тільки своєю потужністю і універсальністю, але і вільним та відкритим вихідним кодом. Це робить Godot доступним для розробників з усього світу та різних рівнів досвіду, незалежно від їхніх фінансових можливостей.

Однією з головних переваг Godot є його мультиплатформеність. Рушій підтримує розробку для різних операційних систем, таких як Windows, macOS та Linux, а також мобільних платформ, таких як Android та iOS. Це дозволяє розробникам створювати ігри, які можуть запускатися на різних пристроях і платформах, розширюючи аудиторію гри та підвищуючи її популярність.

Однією з головних особливостей Godot є інтуїтивно зрозумілий користувацький інтерфейс. Завдяки цьому інтерфейсу розробники можуть швидко освоїти роботу з движком. Крім того, Godot підтримує візуальне програмування за допомогою системи вузлів (node), що спрощує процес розробки для початківців. Скріншот редактора сцени зображений на рисунку 2.1.

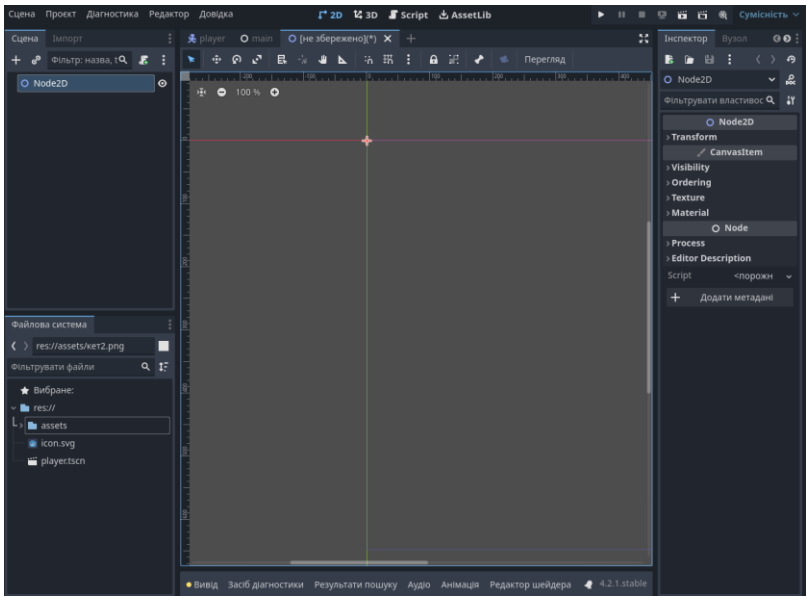



Рис.2.1 Редактор сцени ігрового рушія Godot

Движок підтримує кілька мов програмування, таких як GDScript, C#, Visual Script та C++, що дозволяє розробникам обирати мову, яка найкраще відповідає їхнім потребам та досвіду. Крім того, Godot має велику та активну спільноту розробників, яка постійно працює над вдосконаленням движка та надає підтримку іншим користувачам через форуми, чати та інші ресурси.

Також Godot містить вичерпну документацію та велику кількість навчальних матеріалів, які допомагають розробникам розібратися у всіх особливостях движка та швидко вирішувати будь-які проблеми, що виникають у процесі розробки. Всі ці фактори роблять Godot одним із найкращих варіантів для розробників відеоігор будь-якого рівня.

## 2.2 Мова програмування GDScript

GDScript є основною мовою програмування, використовуваною в Godot для створення ігрової логіки та взаємодій. Мова була розроблена спеціально для Godot та оптимізована для інтеграції з її функціями, що робить її потужним інструментом для розробників усіх рівнів.

Однією з найбільших переваг GDScript є його простота та читабельність. Він має синтаксис, подібний до Python, що полегшує його вивчення та використання. Завдяки цьому навіть новачки в програмуванні можуть швидко освоїти основи і приступити до створення власних ігор. Код на GDScript легко читається і пишеться, що сприяє швидкій розробці та тестуванню ігрових проектів.

GDScript глибоко інтегрований з інтерфейсом та функціями Godot, що дозволяє розробникам повною мірою використовувати можливості рушія. Ця мова дозволяє швидко отримувати доступ до об'єктів сцени, задавати їх властивості та обробляти події. Це спрощує процес написання коду для створення ігрової логіки та дозволяє розробникам легко реалізовувати складні ігрові механіки. Редактор коду Godot можна побачити на рисунку 2.2.

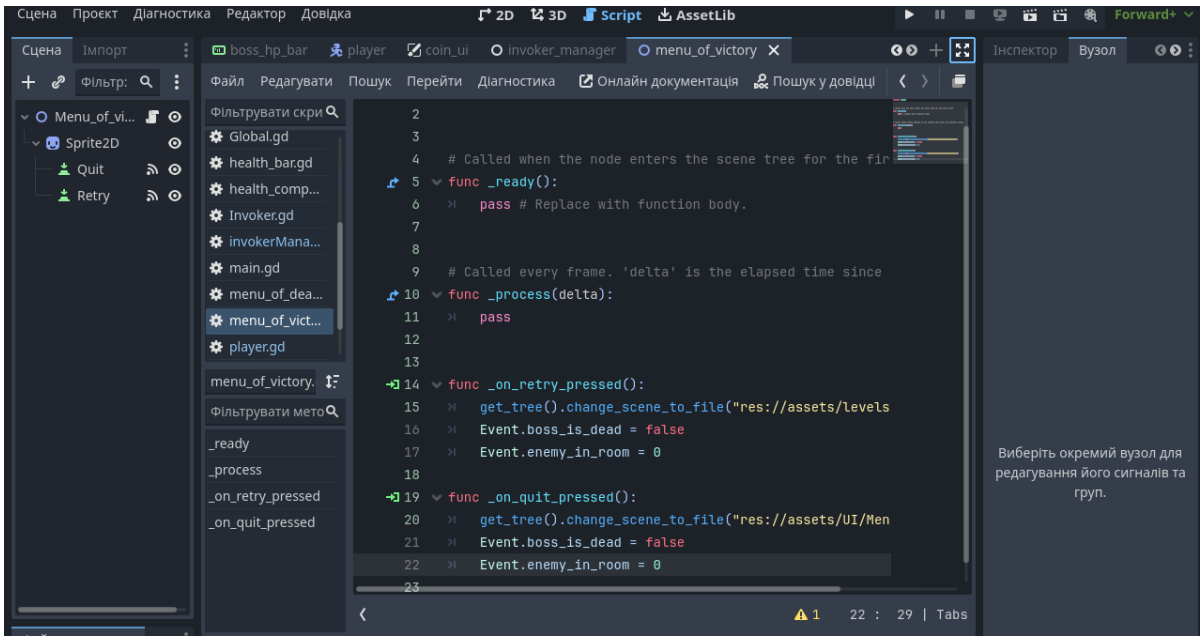



Рис.2.2 Редактор коду Godot з прикладами GDScript

GDScript підтримує принципи об'єктно-орієнтованого програмування (ООП), дозволяючи розробникам структурувати свій код більш організованим та логічним способом. Класи, успадкування та поліморфізм дозволяють створювати масштабовані та прості в обслуговуванні ігрові проекти. Підхід ООП також полегшує повторне використання коду, скорочує час розробки та підвищує ефективність.

Спільнота Godot активно підтримує розробників, надаючи велику кількість навчальних матеріалів та прикладів коду на GDScript. Існує безліч відеоуроків, статей, форумів та інших ресурсів, де ви можете знайти допомогу та поради. Це значно спрощує процес навчання та вирішує проблеми, з якими розробники можуть зіткнутися під час використання GDScript.

Godot надає потужні інструменти для налагодження та профілювання коду на GDScript, що дозволяє розробникам швидко знаходити та виправляти помилки та максимізувати продуктивність гри. Це включає вбудований налагоджувач, можливість встановлення точок зупинки, перевірку значень змінних під час виконання та інші корисні функції.

## 2.3 Графічний редактор Aseprite

Aseprite - це добре відомий інструмент піксельного мистецтва та анімації, який широко використовується розробниками ігор та художниками. Це програмне забезпечення поєднує в собі простоту використання і функціональність, що робить його ідеальним вибором як для початківців, так і для досвідчених фахівців в області графічного дизайну.

Інтерфейс Aseprite інтуїтивно зрозумілий, що полегшує процес його освоєння. Основні інструменти для малювання, анімації та редагування зображень легко доступні, що дозволяє швидко та ефективно створювати піксельну графіку.

Цей редактор пропонує різноманітні можливості для роботи з піксельною графікою. У нього є широкий набір інструментів, включаючи різноманітні пензлі, шари, фільтри та редактори анімації, що дозволяє створювати вражаючі ефекти та анімацію.

Окрім того, Aseprite має розширену палітру кольорів, яка включає в себе багато варіантів кольорів для використання. Користувачі також можуть створювати свої власні кольорові палітри, щоб відповідати їхнім унікальним потребам та стилю.

Aseprite дозволяє розробникам створювати анімовані спрайти для персонажів і об'єктів у грі. Це означає, що ви можете легко створювати різні анімаційні стани для персонажів, такі як ходьба, стрибки, атаки та інші, а також анімовані рухи для об'єктів, таких як вороги, предмети та ефекти. Це дозволяє створювати живу та динамічну графіку, яка зробить вашу гру більш цікавою та захопливою для гравців.

На рисунку 2.3 можна побачити  інтерфейс графічного редактора.

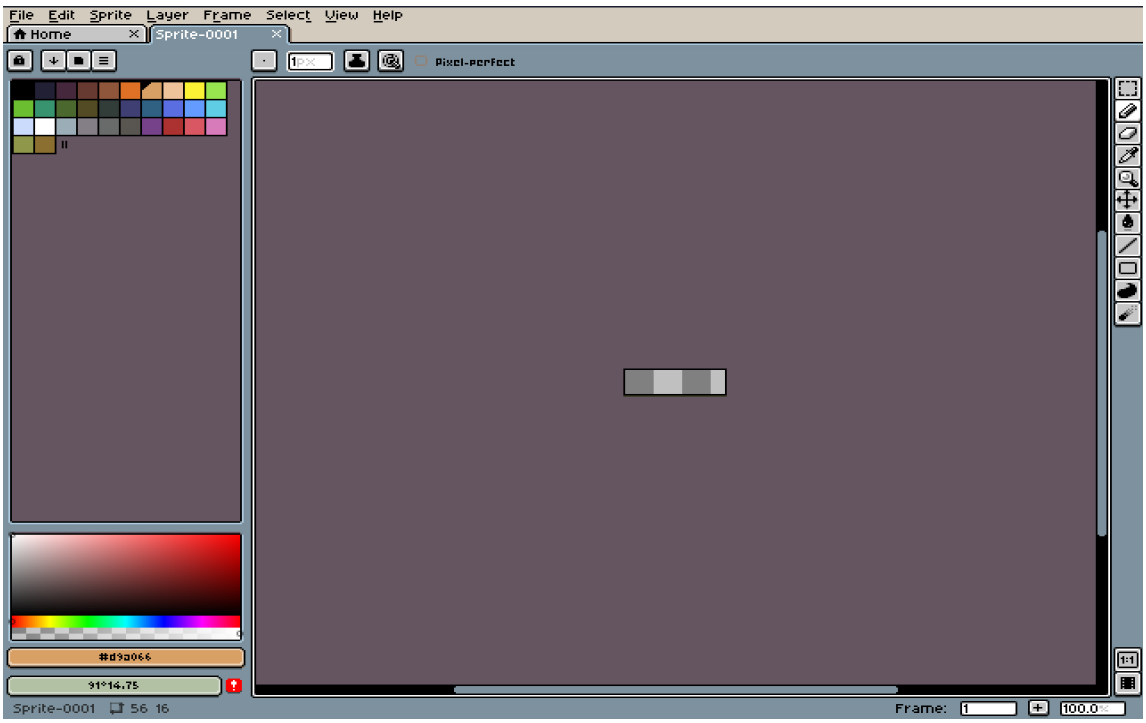



Рис.2.3 Інтерфейс графічного редактора Aseprite

# 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ГРИ “CARD RISK”

## 3.1 Концепт гри “Card Risk”

Концепт гри - важливий етап у розробці гри. На цьому етапі розробники формують основні ідеї та концепти гри, визначають її жанр, цільову аудиторію і базові механіки. Основна мета цієї концепції - чітко визначити, якою має бути гра та який досвід користувачів вона повинна забезпечити

Концепція гри "Card Risk" передбачає проходження рівня, що складається з кімнат, що генеруються випадковим чином. У цих кімнатах гравець зіткнеться з різними ворогами, яких йому потрібно перемогти. Під час боротьби з ворогами або після їхньої поразки, гравець може отримати карту, яка надасть йому певний ефект або можливість. Гра завершується, коли гравець перемагає боса, який є останнім ворогом на рівні.

Цільова аудиторія гри "Card Risk" включає в себе гравців з різним рівнем ігрового досвіду, проте з особливим акцентом на початківців. Ці люди можуть бути новачками у світі відеоігор, або вони можуть бути досвідченими гравцями, які шукають простий та розважальний досвід. Головний привабливий момент для цієї аудиторії - це можливість відпочити та розважитися під час гри, не занадто заглиблюючись у складні механіки або стратегії. Гра "Card Risk" пропонує їм можливість отримати задоволення від гри, не витрачаючи багато часу на вивчення складних правил або глибоких стратегій.

## 3.2 Вимоги до програмного забезпечення

Вимоги до програмного забезпечення є важливим етапом у процесі розробки ігор. Це допомагає розробникам чітко визначити, якими функціями та властивостями повинна володіти гра, щоб відповідати очікуванням користувачів та забезпечувати необхідний їм ігровий досвід. Чітко визначені вимоги знижують

ризик того, що кінцевий продукт не буде відповідати своєму первісному задуму, і полегшують процес тестування і виявлення помилок. Вимоги зазвичай поділяються на функціональні та нефункціональні.

Функціональні вимоги точно описують, що повинна робити гра, які її основні характеристики та функції. Вони включають такі аспекти, як ігрова механіка, поведінка персонажа, взаємодія користувача з грою та інші важливі елементи. Функціональні вимоги гарантують, що гра буде працювати відповідно до очікувань гравця і забезпечить передбачуваний ігровий досвід.

Нефункціональні вимоги визначають якісні характеристики гри: продуктивність, надійність, простоту використання, сумісність з різними платформами. Вони однаково важливі, оскільки забезпечують стабільність і задоволення від гри для кінцевого користувача. Нефункціональні вимоги гарантують, що гра не тільки функціональна, але й портативна та надійна для гравця.

### 1.2.1 Функціональні вимоги

Функціональні вимоги:

1. Можливість почати та закінчити гру.

1.1 Гра починається при натисканні кнопки “Play” .

1.2 Гра може бути завершена на будь-якому етапі геймплею або при досягненні кінцевої мети (Перемога над босом).

2. Реалізація логіки відкриття та закриття дверей кімнат рівня.

2.1 Коли гравець входить в кімнату двері зачиняються.

2.2 При вбивстві всіх ворогів на рівні двері відчиняються, дозволяючи гравцю далі проходити рівень.

3. Забезпечити можливість переміщуватися між різними кімнатами гри.

3.1 Гравець може переміщатися між кімнатами, які з'єднані дверима, за допомогою відповідних клавіш

3.2 Перехід між кімнатами супроводжується плавним зміщенням камери для максимально комфортного сприйняття гравцем оточуючого середовища.

4. Реалізація різних ефектів для кожного з виду карт:

4.1 Черви - Збільшують швидкість героя.

4.2 Піки - Збільшують дальність атаки героя.

4.3 Трефи - Збільшують здоров'я гравця.

4.4 Бубни - Збільшують урон атаки героя.

5. Реалізація різних типів ворогів:

5.1 Привид, що постійно слідкує за гравцем.

5.2 Привид-стрілець, що не рухається, але вистрілює в гравця пулею.

6. Реалізація боса.

6.1 Бос атака, якого призиває додаткових привидів для атаки гравця.

### 3.2.2 Нефункціональні вимоги

Нефункціональні вимоги:

1. Процедурне генерування рівнів.

1.1 Гра повинна використовувати алгоритми для генерації унікальних і цікавих рівнів при кожному новому запуску гри. Це забезпечить більшу різноманітність в геймплеї.

2. Підтримка операційної системи Windows.

2.1Гра повинна безперервно працювати на різних версіях операційної системи Windows, включаючи Windows 7, Windows 8, та Windows 10. Підтримка широкого спектру версій операційної системи дозволяє більшому колу користувачів насолоджуватися грою, незалежно від їх поточної конфігурації ПК

2.2 Гра повинна ефективно використовувати можливості, які пропонує кожна конкретна версія операційної системи Windows. Оптимізація ресурсів системи, таких як процесор, оперативна пам'ять та графічний прискорювач, дозволить забезпечити стабільну та оптимальну роботу гри на будь-якому обладнанні.

3. Управління за допомогою клавіатури.

3.1 Управління героєм повинно бути за допомогою клавіатури.

4. Анімовані рухи героя.

4.1 Рухи головного героя повинні бути анімованими, що додасть більшу реалістичність та живість грі.

### 3.3 Use case діаграма

Діаграма випадків використання - це одна з діаграм уніфікованої мови моделювання (UML), яка представляє взаємодію між учасником (користувачем або зовнішньою системою) та системою, про яку йде мова, для досягнення певної мети. Вона надає загальне уявлення про функціональність системи, пояснюючи різні способи взаємодії користувача з нею.

Позначення UML надає візуальну мову, яка дозволяє розробникам програмного забезпечення, дизайнерам та іншим зацікавленим сторонам обмінюватися інформацією та документувати архітектуру та поведінку системи послідовно та зрозуміло.

Основні елементи діаграми варыантів використання:

1. Актор: користувач або зовнішня система, що взаємодіє з системним суб'єктом, може бути основним (primary), який ініціює взаємодію, і вторинним (auxiliary), який забезпечує допоміжну функціональність.

2. Варіант використання: конкретний сценарій, що описує взаємодію між суб'єктом та системою для досягнення певної мети. Варіанти використання включають назву, мету, основні напрямки дій та альтернативні потоки дій.

3. Система: Межа, що відокремлює внутрішню частину системи від зовнішніх акторів. Це показує, які частини функціональності належать до системи, а які — до зовнішніх елементів.

4. Асоціації: Лінії, що з'єднують акторів із випадками використання, показуючи, які дії може виконувати актор. Відображають прямий зв'язок між актором та випадком використання.

Відношення:

4.1 Include: Один варіант використання завжди включає інший.

4.2 Extend: Один варіант використання може бути розширений іншим за певних умов.

4.3 Generalization: Один варіант використання може бути спеціалізацією іншого

На рисунку 3.1 зображена діаграма варіантів використаня гравця де можна побачсити основні механіки які доступні гравцеві. Такі як:

1. Почати гру.
2. Закінчити гру.
3. Призупинити гру.
4. Керування рухом героя ,що включає в себеможливість проходичи в інші кімнати через двері.
5. Атака ворогів.
6. Підібрати карту, що включає в себе отримання ефекту на основі підібраної карти.

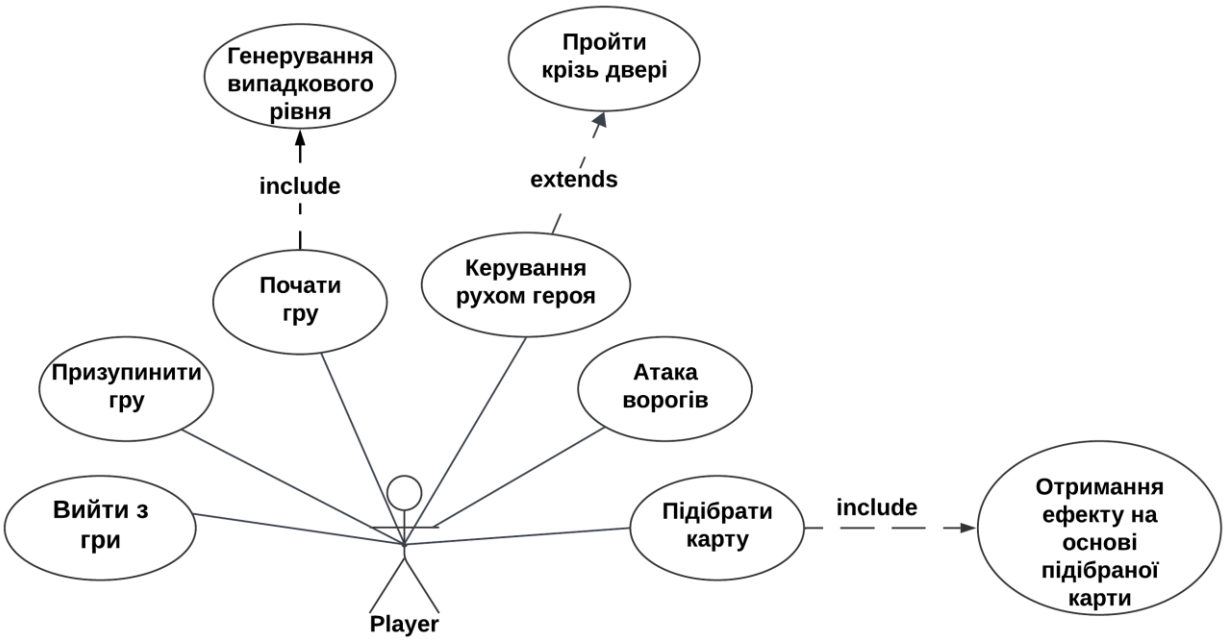



Рис.3.1 Use case діаграма гравця

### 3.4 Розробка діаграми класів

Діаграми класів-одна з ключових діаграм в уніфікованій мові моделювання (UML), що використовується для візуалізації структури програмного забезпечення.1 вона відображає класи програм, їх атрибути, методи, взаємозв'язки між ними та інші важливі аспекти системи. Розглянемо його компоненти та роль у розробці програмного забезпечення:

Класи є важливим компонентом діаграм класів, що є абстракцією об'єктів або типів даних, які мають спільні властивості та поведінку. Вони служать основою для моделювання структури програмного забезпечення та визначення його компонентів.

Кожен клас може мати унікальну назву, яка ідентифікує його в системі. Крім того, у класі можуть бути оголошені атрибути та методи. Атрибути визначають властивості або дані, що належать до класу, А методи визначають дії, які можуть виконуватися з цими атрибутами.

Класи можуть бути пов'язані відносинами, такими як асоціації, конфігурації, агрегації та узагальнення. Ці зв'язки вказують на взаємозв'язки між різними класами та допомагають визначити взаємодію між ними.

На діаграмі класів клас представлений у вигляді прямокутника, в якому вказано назву класу. Також можуть бути вказані атрибути та методи класу. Кожен клас має свою внутрішню структуру і може бути пов'язаний з іншими класами в системі.

На діаграмі класів:

- Клас представлений у вигляді прямокутника з ім'ям класу всередині.
- У прямокутнику також можуть бути вказані атрибути та методи класу.
- Кожен клас може мати свою внутрішню структуру та бути пов'язаним з іншими класами в системі.

Діаграма класів допомагає розробникам отримати загальне уявлення про структуру програми та взаємозв'язки між її складовими частинами, що сприяє зрозумінню та розвитку програмного забезпечення.

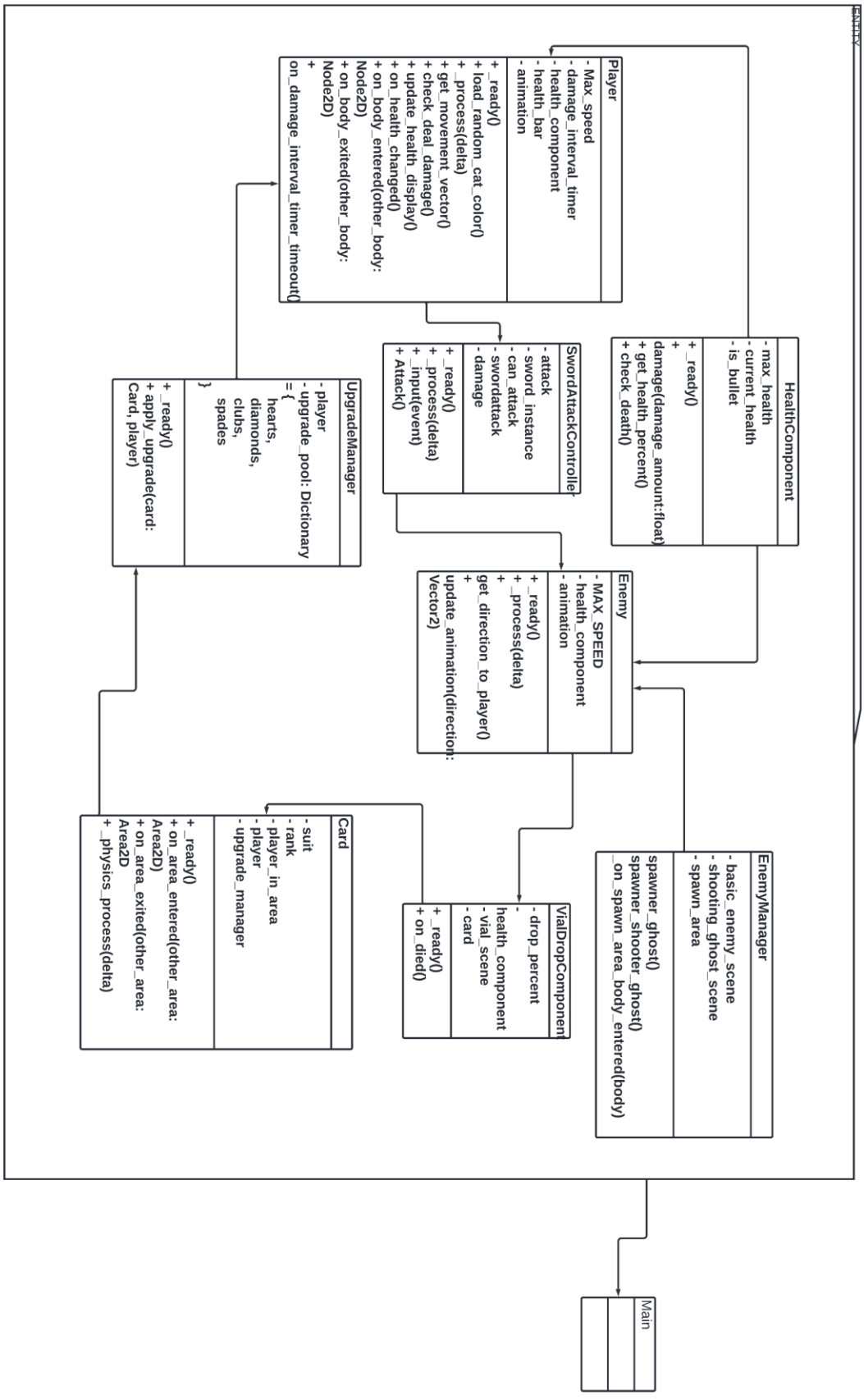



Рис.3.2 Діаграма класів сутностей

На рисунку 3.2 можна побачити діаграму класів сутностей, де можна виділити наступні головні класи:

HealthComponent – відповідає за здоров’я гравця та ворогів, їх взаємодію один з одним, випадіння предметів з ворогів.

Card – відповідає за створення карт у грі.

UpgradeManager – відповідає за надання ефекту гравцеві на основі масті та рангу карти. Він відстежує та керує покращеннями, які гравець може отримати під час гри.

Також була розроблена діаграма класів кімнати та боса, яка зображена на рисунку 3.3.

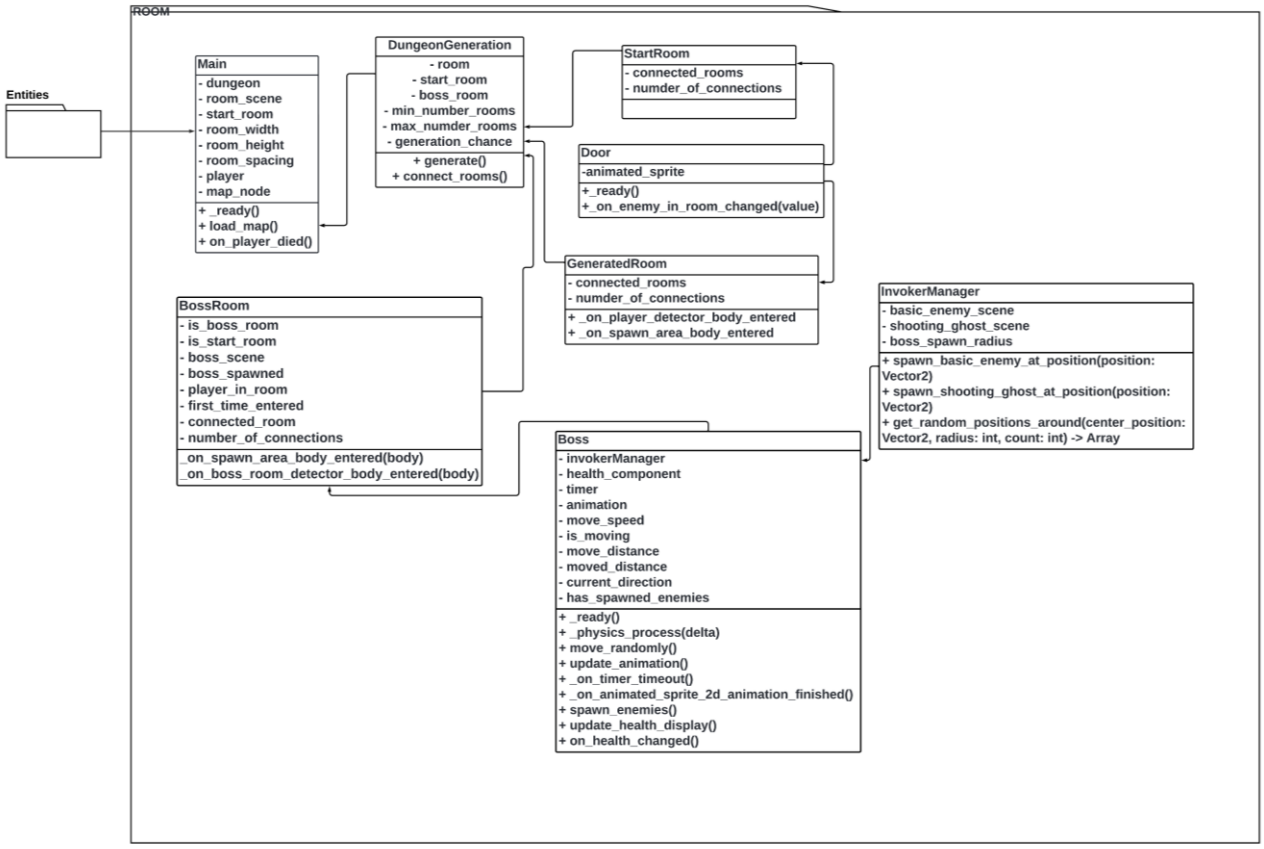



Рис.3.3 Діаграма класів рівня та боса

Тут можна побачити такі головні класи:

DungeonGeneration – відповідає за процедурну генерацію рівня.

Boss – відповідає за характеристики боса.

InvokerManager – відповідає за головну атаку (призив привидів для додаткової атаки гравця).

Також була розроблена діаграма діяльності ігрового процесу яка зображена на рисунку 3.4.

Ця діаграма описує послідовність дій, які виконуються під час гри, включаючи взаємодії між гравцем, ворогами та різними об'єктами в грі. Вона допомагає зрозуміти, як різні компоненти системи взаємодіють один з одним для створення динамічного ігрового досвіду.

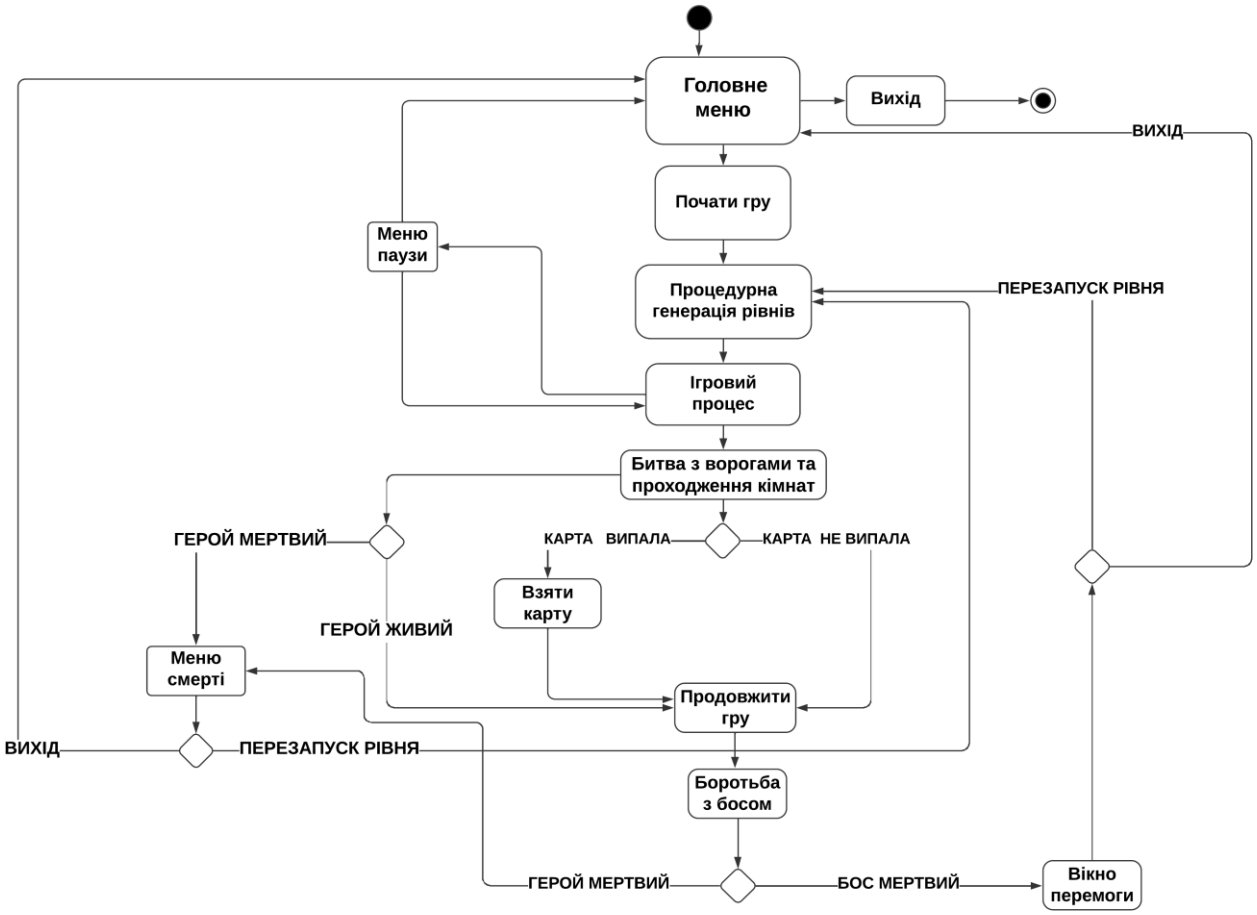



Рис.3.4 Діаграма діяльності ігрового процесу

## 3.5 Створення 2D графіки для гри

2D-графіка є важливим аспектом гри, оскільки вона визначає візуальний стиль і сприйняття гравцем. Візуальні ефекти не тільки роблять гру привабливою, але і допомагають передати настрій, атмосферу і функціональність ігрового процесу. Високоякісна 2D-графіка значно покращує ігровий процес і дає гравцям можливість повністю зануритися в ігровий світ

### 3.5.1 Дизайн героя, боса та ворогів

Дизайн персонажів є однією з найважливіших частин 2D-графіки у грі. Вони є основними елементами, з якими гравець взаємодіє протягом всього ігрового процесу. Розглянемо детальніше процес створення дизайну для головних персонажів гри: героя, боса та ворогів.

#### 3.5.1.1 Головний герой

Розробка концепт-арту починається з визначення зовнішнього вигляду героя, його стилю, одягу, озброєння та інших атрибутів. На цьому етапі важливо створити унікальний та привабливий образ, який відображатиме характер і роль героя в грі.

У моєму випадку концепт-арт головного героя представлений милим котиком. Він має простий, але чарівний дизайн, який легко запам'ятовується і привертає увагу гравців. Котик ходить по кімнатах і бореться з ворогами, використовуючи ефекти які він отримав при підбиранні карт.

Також в грі є три різних варіантів  вигляду головного героя ,який обирається при початку гри випадково. Варіанти головного героя представлені на рисунку 3.5.

Рис.3.5 Варіанти вигляду головного героя

Після затвердження концепт-арту створюються спрайти героя. Кожен спрайт представляє один кадр анімації для різних дій героя, таких як ходьба, біг, стрибки, атаки та отримання пошкоджень. Спрайти об'єднуються у спрайт-листи. На рисунку 3.6 можна побачити намальовані анімації для руху героя у всі сторони.

Рис.3.6 Спрайт-лист головного героя (Сірий)

### 3.5.1.2 Бос

Концепт-арт включає деталі зовнішності боса, його розмір, озброєння та особливі риси, які підкреслюють його силу та небезпеку. Бос представлений у вигляді чоловіка з нездоровим виглядом, а його одяг прикрашений хрестами та зображеннями черепів, що робить його образ трохи моторошним і вражаючим. Ці деталі додають характеру та інтриги його взаємодії з гравцем і підкреслюють його роль як потужного та загрозливого противника в грі. Спрайти з анімації ходьби боса зображено на рисунку 3.7.

Рис.3.7 Спрайт - лист боса

Також головним етапом це є розробити анімацію перед атакою боса, основна атака даного боса є призив привидів. Спрайти призиву привидів зображено на рисунку 3.8.

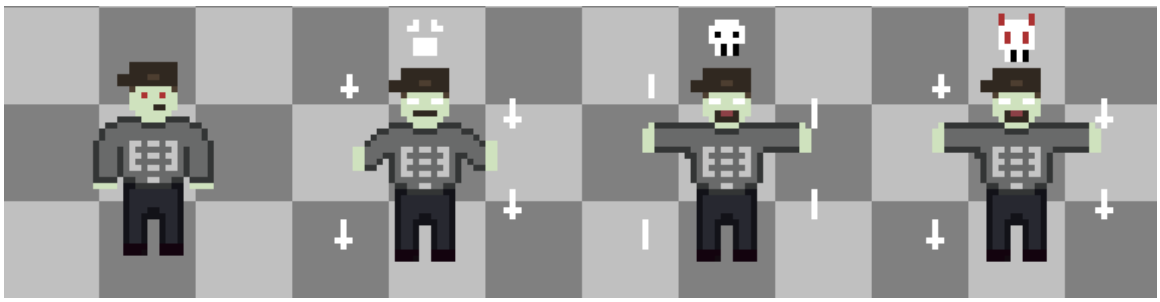


Рис.3.8 Спрайт - лист призиву привидів босом

### 3.5.1.3 Вороги

У грі присутні два варіанти ворогів.

Привид: Цей ворог має вигляд містичного спіриту, який переслідує гравця. Він має прозоре тіло з відблисками світла, нагадуючи про свою нематеріальність.

Привид - стрілець: Цей ворог має такий же зовнішній вигляд, як і звичайний привид, але він має вуха кота. Він не переслідує гравця, але стоїть на місці та стріляє в нього. Виглядає він як комбінація містичного привида з характерними ознаками кота, що робить його ще більш загадковим та цікавим супротивником.

На рисунку 3.9 можна побачити спрайти ворогів.

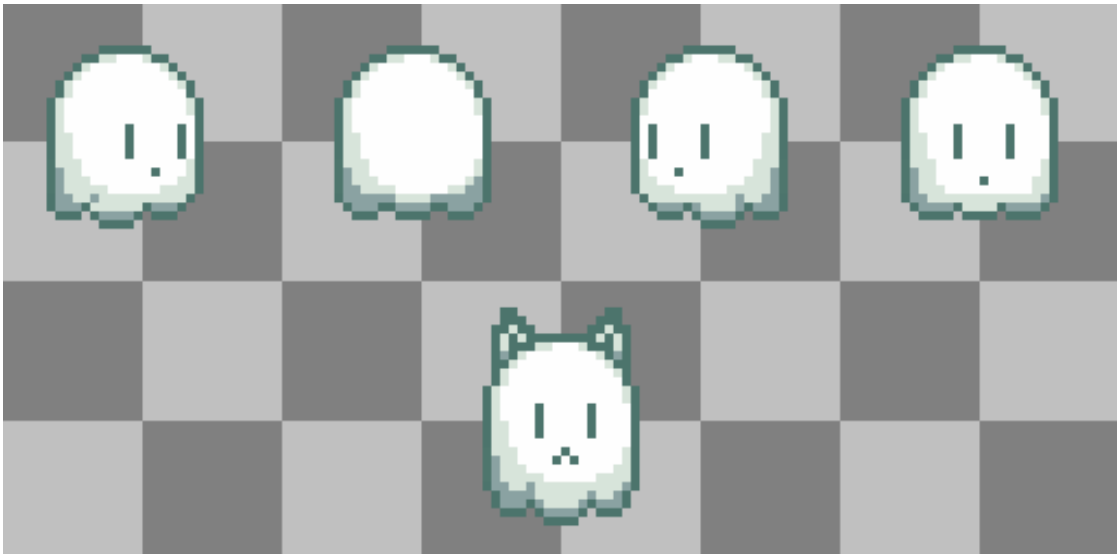


Рис.3.9 Спрайти ворогів

### 3.5.2 Дизайн кімнат

Дизайн кімнат - це ще один важливий аспект гри, який визначає візуальну привабливість та ігровий досвід. В даній грі кімнати мають вигляд, як кімнати замку ,але мають однотонність. Спрайт кімнати можна побачити на рисунку 3.10.

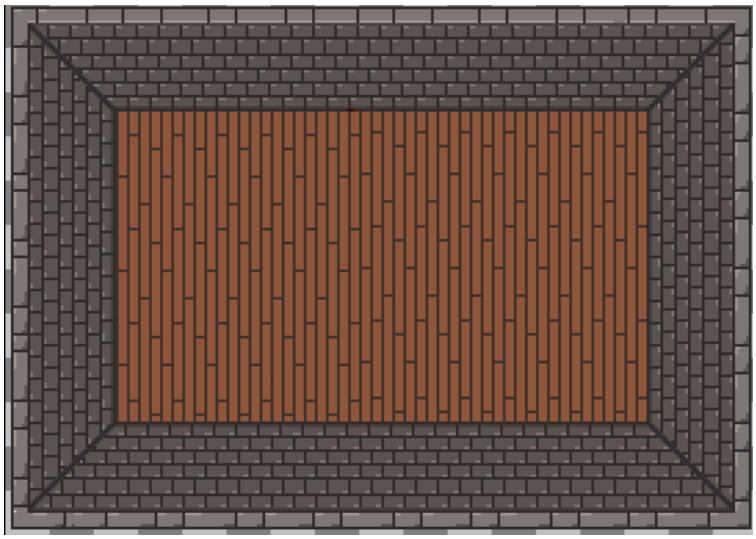


Рис.3.10 Спрайт кімнати

#### 3.5.2.1 Дизайн дверей

Дизайн дверей також є важливою деталлю в розробці гри в жанрі “Roguelike” В даній грі дизайн є простим, але він ідеально вписується в атмосферу та загальний стиль гри. Спрайти зачинених та відчнинених дверей можна побачити на рисунку 3.11.

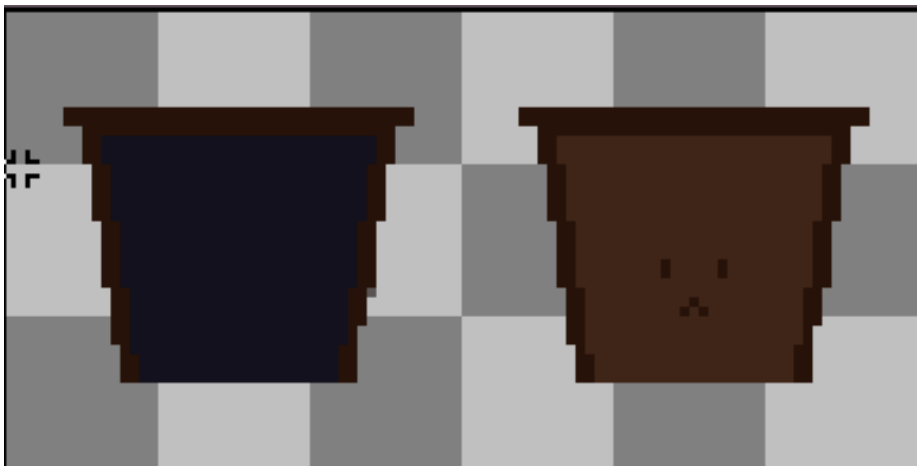


Рис.3.11 Спрайти зачинених та відчинених дверей

### 3.5.3 Дизайн карт

Дизайн карт - це важлива складова розробки гри, яка визначає зовнішній вигляд ігрових карт, їхню структуру, естетику та функціональність. В грі "Card Risk" дизайн карт відіграє ключову роль у візуальному представленні гри, впливаючи на загальний ігровий досвід гравців. Спрайт карт зображено на рисунку 3.12.

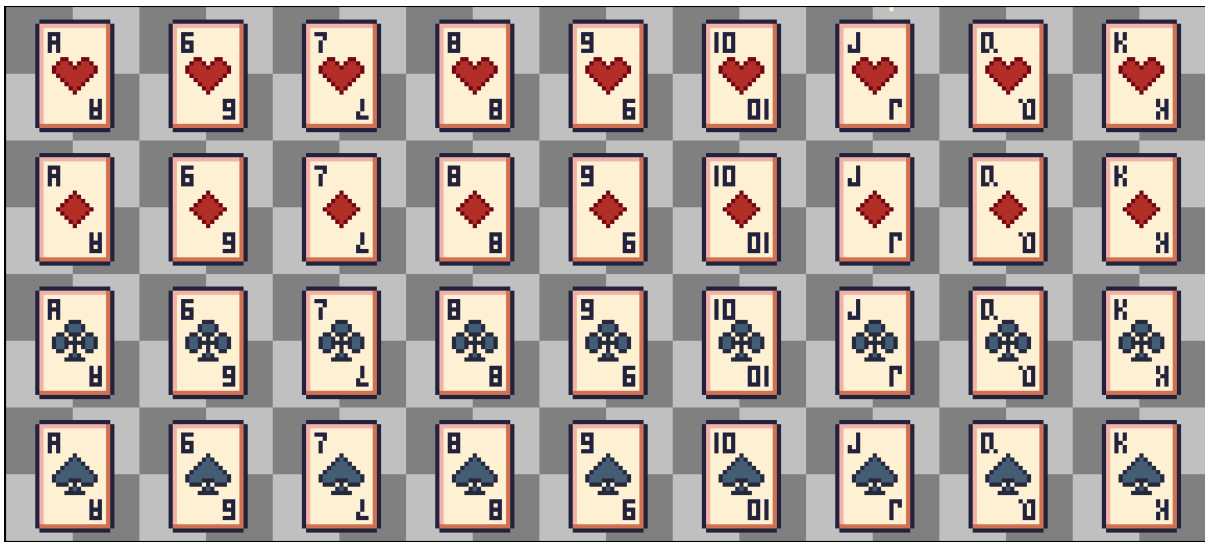


Рис.3.12 Спрайт карт

## 3.6 Реалізація процедурного генерування

Процедурне генерування - це метод створення контенту у вигляді ігрових рівнів, мап, об'єктів та іншого, що використовує алгоритми та випадковість для автоматичного створення різноманітних ігрових елементів. У жанрі "Roguelike" процедурне генерування використовується для створення лабіринтоподібних рівнів, які гравець досліджує під час гри.

У цій роботі процедурне покоління починається з початкової кімнати, яка є координатою (0;0), яка є центральною точкою Згенерованого лабіринту. Кожна нова кімната додається з випадковою ймовірністю в певному напрямку від поточної кімнати. Після додавання кімнати з'єднайте її з існуючою кімнатою через

коридор або прохід. Цей процес триває до тих пір, поки не буде досягнуто задану кількість кімнат. На  рисунку 3.13 можна побачити приклад згенерованого рівня.

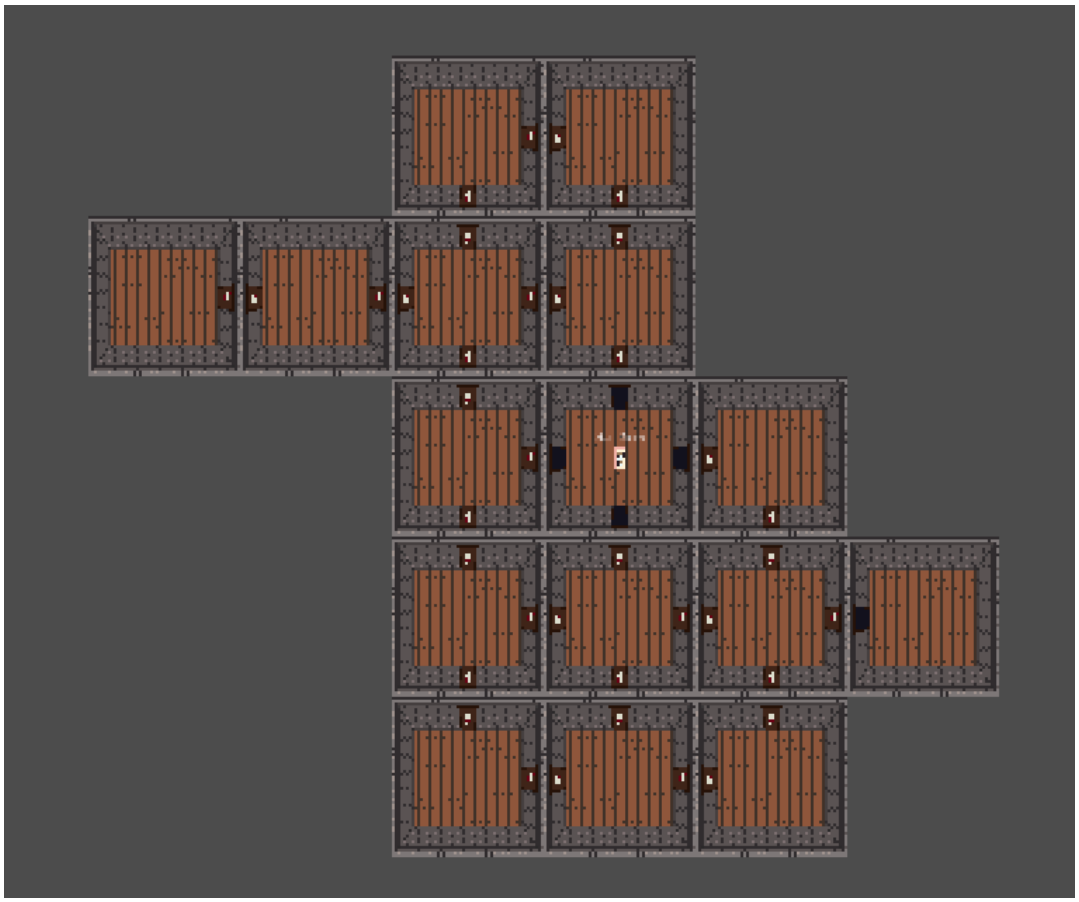


Рис.3.13 Приклад згенерованого рівня процедурною генерацією

# 4 ТЕСТУВАННЯ ГРИ “CARD RISK”

## 4.1 Типи тестування

Тестування програмного забезпечення-важливий етап розробки, який дозволяє виявляти і виправляти помилки, покращувати якість продукту і забезпечувати стабільну роботу. В основному існує кілька типів тестів для тестування ігор, кожен з яких відіграє певну роль у забезпеченні якості:

1. Функціональне тестування - Функціональне тестування спрямоване на перевірку відповідності гри встановленим вимогам і специфікаціям. Тест охоплює всі аспекти ігрового процесу, включаючи базову ігрову механіку, правила, систему підрахунку очок, елементи керування, меню та інтерфейс користувача.

2. Нефункціональне тестування - Нефункціональне тестування передбачає оцінку таких характеристик гри, як продуктивність, надійність, масштабованість та простота використання. Мета цього тесту-переконатися в тому, що гра працює ефективно і стабільно в різних умовах. Продуктивність гри вимірюється часом завантаження, частотою кадрів, використанням пам'яті і процесора, надійність перевіряється в ході тривалих ігрових сесій для виявлення можливих збоїв і витоків пам'яті. Масштабованість оцінюється шляхом тестування гри на різних апаратних конфігураціях і операційних системах, щоб переконатися, що вона коректно працює на різних платформах.

3. Тестування сумісності - Цей тест спрямований на перевірку сумісності гри з різними конфігураціями апаратного та програмного забезпечення. Тестувальники перевіряють, чи буде гра працювати з різними операційними системами, версіями драйверів, відеокартами та процесорами. Вони також перевіряють сумісність гри з різними периферійними пристроями, такими як геймпади, клавіатури, миші та гарнітури віртуальної реальності.

4. Тестування локалізації - Тести локалізації включають перевірку точності перекладів, адаптацію тексту, графіки та інших елементів гри до різних мов і

культур. Тестувальники перевіряють, чи правильно відображається весь текст, чи відповідає переклад оригінальному вмісту, а також чи немає текстових помилок або неправильних символів. Вони також перевіряють, чи адаптовані графічні елементи та звуки до культурних особливостей різних країн.

5. Альфа та бета тестування - Є важливим етапом в розробці гри і проводиться до офіційного релізу. Альфа-тестування проводиться на ранніх стадіях розробки, щоб виявити основні недоліки та проблеми гри. Зазвичай це тестування проводиться внутрішньою командою розробників. Бета-тестування проходить на різних етапах, включаючи тестування гри зовнішніми користувачами (гравцями). Воно допомагає виявити проблеми, які, можливо, були упущені під час внутрішнього тестування, і надає реальні відгуки користувачів про ігровий процес і загальну якість гри.

### 4.2 Особливості тестування ігор

Тестування ігор має свої особливості, які відрізняють його від інших видів тестування програмного забезпечення. Ці особливості пов'язані з характером гри, її інтерактивністю, графічними компонентами та різними платформами, на яких вони можуть працювати. Ось основні аспекти, які слід враховувати при тестуванні гри:

1. Інтерактивність та ігровий процес - Гра являє собою інтерактивний продукт, в якому користувач активно взаємодіє з програмою. Це означає, що Тестувальник повинен ретельно перевірити всі можливі сценарії ігрового процесу, включаючи різні дії гравця, можливі комбінації команд і реакцію гри на ці дії. Особливу увагу слід приділити тестуванню базової механіки системи управління mushroom і відповідності реакції гри на дії гравця очікуваному результату.

2. Графічні компоненти - Оскільки графіка є важливим елементом гри, тести включають в себе перевірку якості зображення, коректного відображення текстур, анімації, світлових ефектів і тіней. Важливо протестувати гру з різними

налаштуваннями графіки та різною роздільною здатністю екрана, щоб переконатися, що вона працює належним чином на різних пристроях.

3. Звуковий супровід - Звук відіграє важливу роль в ігровому процесі, тому тестування повинно включати перевірку якості звуку, правильності відтворення звукових ефектів, музики та голосових команд. Необхідно переконатися, що звук синхронізований з діями на екрані і відповідає ігровим подіям.

### 4.3 Тестові сценарії гри “Card risk”

"Перемога" у даному сценарії перевіряється на те, чи вдалося гравцеві перемогти боса. Сценарій охоплює ситуацію, коли гравець потрапляє до кімнати боса і змушений взаємодіяти з ним, а також перевіряється, чи викликається вікно перемоги в той момент, коли шкала життя боса досягає нуля.

“Смерть” у даному сценарії перевіряється  перевіряється, чи правильно викликається меню смерті, коли гравець досягає нульового рівня здоров'я під час взаємодії з ворогами

“Перезапуск рівня” у даному сценарії перевіряється чи перезапускається рівень з меню смерті. У сценарії розглядається ситуація коли гравець при проходженні рівня помирає та з вікна смерті обирає кнопку “Retry”.

“Ефект карти - Чірва” у даному сценарії перевіряється, чи правильно видається гравцеві ефект при підбиранні карти. Сценарій описує ситуацію, коли гравець убиває ворога, і з нього випадає карта. Після того, як гравець підбирає цю карту, вона має надавати прискорення ,сила ефекта залежить від рангу карти .

Тестовий сценарій - " Перемога "

Кроки:

1. Гра починається з головного меню.
2. Гравець з’являється в стартовій кімнаті.
3. Гравець успішно проходить кімнати.
4. Гравець знаходить кімнату боса.
5. Гравець вбиває боса.

6. Викликається вікно перемоги.

Цей тестовий сценарій допомагає перевірити коректність перевірки смерті боса та при досяганні цієї умови виклику вікна перемоги

Тестовий сценарій - "Смерть"

Кроки:

1. Гра починається з головного меню.
2. Гравець з’являється в стартовій кімнаті.
3. При проходженні рівня шкала здоров’я героя рівна нулю.
4. Викликається вікно смерті

Цей тестовий сценарій допомагає перевірити коректність смерті головного героя  та при досяганні цієї умови виклику вікна смерті.

Тестовий сценарій - "Перезапуск рівня"

Кроки:

1. Гра починається з головного меню.
2. Гравець з’являється в стартовій кімнаті.
3. При проходженні рівня шкала здоров’я героя рівна нулю.
4. Викликається вікно смерті.
5. Гравець нажимає на кнопку “Retry”.
6. Перезапускається сцена з рівнем.

Цей тестовий сценарій допомагає перевірити коректність роботи кнопки “Retry” в меню смерті, яка повинна перезапускати рівень.

Тестовий сценарій - " Ефект карти - черви "

Кроки:

1. Гра починається з головного меню.
2. Гравець з’являється в стартовій кімнаті.
3. При проходженні рівня з ворога  випала карта.
4. Гравець підбирає карту з мастю черви.
5. У гравця збільшилась швидкість (сила ефекту залежить від рангу карти).

Цей тестовий сценарій допомагає перевірити коректність  надання ефекту гравцеві від карти з мастю черви.

# ВИСНОВКИ

1. Проведено аналіз існуючих ігор в жанрі "Roguelike", що дозволило визначити їхні основні особливості та ігрові механіки.

2. Визначено концепт гри, особливості геймплею та вимоги до розробки гри в жанрі “Roguelike”. В якості ключових вимог визначено такі як перманентна смерть та процедурне генерування рівнів. Для покращення геймплею концепт гри включає появу випадкових карт для посилення сил персонажу.

3. Розроблено алгоритми, що забезпечують реалізацію вимог, в тому числі, алгоритми для перманентної смерті та процедурного генерування рівнів.

4. Створено основні 2D об`єкти гри, такі як кімнати, вороги та головний герой з виокристанням програмному  забезпеченню “Aseprite”. Основні ігрові механіки, такі як рух героя, підбирання карт та пересування між кімнатами, реалізовані за допомогою ігрового рушія Godot та мови програмування GDScript.

5. Розроблено тестові сценарії для тестування гри «Card risk», які включають кейси тестування інтерактивності та ігрового процесу, тестування графічних компонентів та звукового супроводу. Всі тестові сценарії завершено з позитивним результатом.

Робота пройшла апробацію на наукових конференціях, за результатами апробації опубліковано тези доповідей:

1. Куріс А.С., Золотухіна О.А. Визначення типових характеристик гри жанру "Roguelike”. Матеріали IV Всеукраїнської науково-практичної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті».  Збірник тез. 15.05.2024, ДУІКТ, м. Київ. К.:ДУІКТ. С.164-165.

2. Куріс А.С., Золотухіна О.А. Вплив передових технологійна ігрову індустрію. Матеріали V Всеукраїнської науково-практичної конференції молодих вчених та здобувачів вищої освіти присвяченої Дню науки (17 травня 2024 р.). За ред. Г.В. Жосан, Г.О. Димової та ін. Херсон-Кропивницький: Видавництво ФОП Вишемирський В.С., 2024. С. 81-82.

# ПЕРЕЛІК ПОСИЛАНЬ


1. Godot Engine Documentation - GDScript. 2023. URL: https://docs.godotengine.org/en/stable/tutorials/scripting/gdscript/gdscript_basics.html
2. Deep Reinforcement Learning with Godot Game Engine [Електронний ресурс] / Qusay H. M., Mahesh R. Mahesh. – 2024. – Режим доступу до ресурсу: https://www.researchgate.net/publication/378759093_Deep_Reinforcement_Learning_with_Godot_Game_Engine
3. Rogue (video game). (2023, May 14). In Wikipedia, The Free Encyclopedia. Retrieved from https://en.wikipedia.org/wiki/Rogue_(video_game)
4. Куріс А.С., Золотухіна О.А. Визначення типових характеристик гри жанру "Roguelike". Матеріали IV Всеукраїнської науково-практичної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. 15.05.2024, ДУІКТ, м.Київ. К.:ДУІКТ − с.164-165.
5. Куріс А.С., Золотухіна О.А. Вплив передових технологійна ігрову індустрію. Матеріали V Всеукраїнської науково-практичної конференції молодих вчених та здобувачів вищої освіти присвяченої Дню науки (17 травня 2024 р.). За ред. Г.В. Жосан, Г.О. Димової та ін. Херсон-Кропивницький: Видавництво ФОП Вишемирський В.С., 2024. с. 81-82.

# ДОДАТОК А ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ


ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО -КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО -НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ


## Розробка гри Card risk в жанрі Roguelike засобами рушія Godot мовою GDScript

Виконала студентка 4 курсу
групи ПД-42
Куріс Анжеліка Сергіївна
Керівник роботи
К.т.н., доц., доцент кафедри ІПЗ Золотухіна Оксана Анатоліївна

Київ – 2024

## МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** – покращення геймплею для гравців-початківців в жанрі “Roguelike” з використанням ігрового рушія Godot та мови Gdscript.
- **Об’єкт дослідження** – геймплей для гравців-початківців в жанрі “Roguelike” .
- **Предмет дослідження** – гра для гравців-початківців в жанрі “Roguelike” .

2

# ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести аналіз та дослідження існуючих ігор в жанрі “Roguelike”.
2. Визначити концепт гри, особливості геймплею та вимоги до розробки гри в жанрі “Roguelike”.
3. Розробити інтерфейс та візуальне оформлення гри “Card risk”.
4. Розробити та програмно реалізувати ігрові механіки гри в жанрі “Roguelike” за допомогою рушія Godot та мови Gdscript.
5. Провести тестування гри “Card risk”.

3

# АНАЛІЗ АНАЛОГІВ

| **Критерії порівняння** | **The Binding of Isaac** | **Enter the Gungeon** | **Card risk** |
|---|---|---|---|
| Системні вимоги | 2 GB RAM; Intel Core 2 Duo; Discreet video card | 2 GB RAM; Intel Core 2 Duo; GeForce 7600 GS | 2 GB RAM; Intel Core 2 Duo; OpenGL 3.3 |
| Наявність процедурної генерації рівнів | + | + | + |
| Складність геймплею | Складна | Складна | Середня |
| Тривалість ігрової сесії | 15 – 30 хвилин | 15 – 30 хвилин | До 10 хвилин |
| Елемент ризику з використанням гральних карт | - | - | + |
| Інтуїтивно зрозуміле управління | - | + | + |

4

# КОНЦЕПТ ГРИ

- Гравець повинен пройти рівень, який випадково генерується з кімнат.
- Протягом гри головний герой б’ється з ворогами.
- В процесі гри при проходження кімнати може випасти з деяким шансом карта для посилення сил персонажу.
- Гра закінчується при перемозі над босом рівня або при смерті персонажу.



# ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

**Функціональні вимоги:**

1. Можливість почати та закінчити гру (гра не є нескінченою).
2. Реалізувати логіку відкриття та закриття дверей кімнати рівня.
3. Забезпечити можливість переміщуватися між різними кімнатами гри.
4. Реалізувати різні ефекти для кожного з виду карт(черви - збільшують швидкість героя, піки - збільшують дальність атаки героя, трефи - збільшують здоров’я гравця, бубни - збільшують урон атаки героя).
5. Реалізувати різні типи ворогів(привид, що постійно слідкує за гравцем та привид -стрілець,що не рухається та вистрілює в гравця пулею).
6. Реалізувати боса , де його атакою буде призов привидів.

**Нефункціональні вимоги:**

1. Процедурне генерування рівнів.
2. Гра повинна функціонувати під операційною системою Windows.
3. Управління переміщенням персонажа повинно здійснюватись за допомогою клавіатури.
4. Рухи героя повинні бути анімованими.

# ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

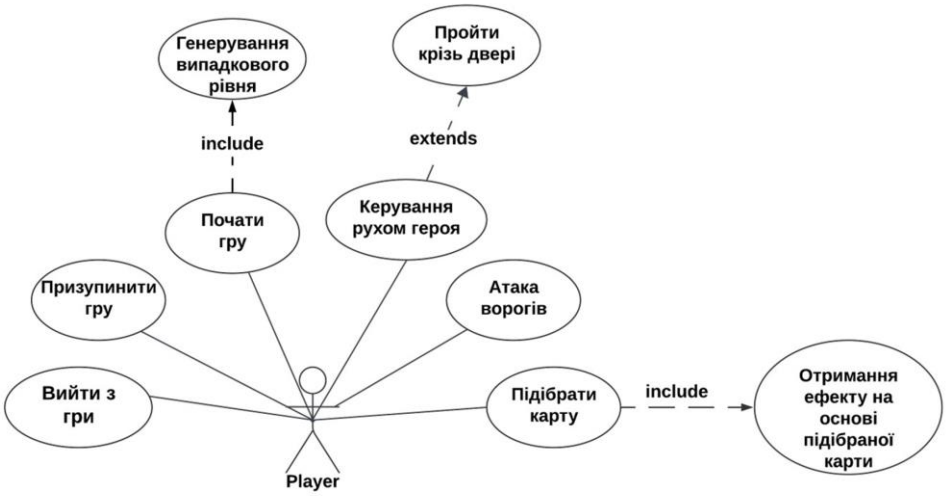
Aseprite

GODOT

GDScript

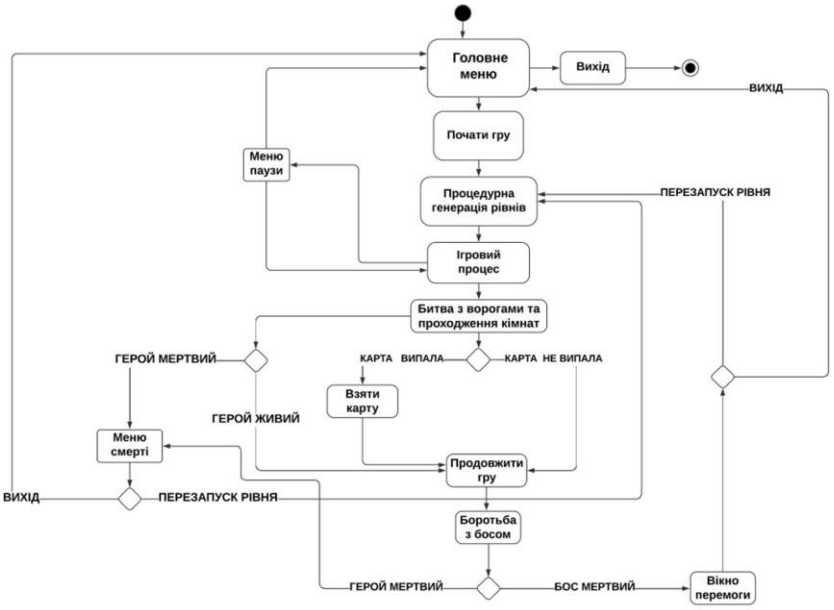
7

# ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



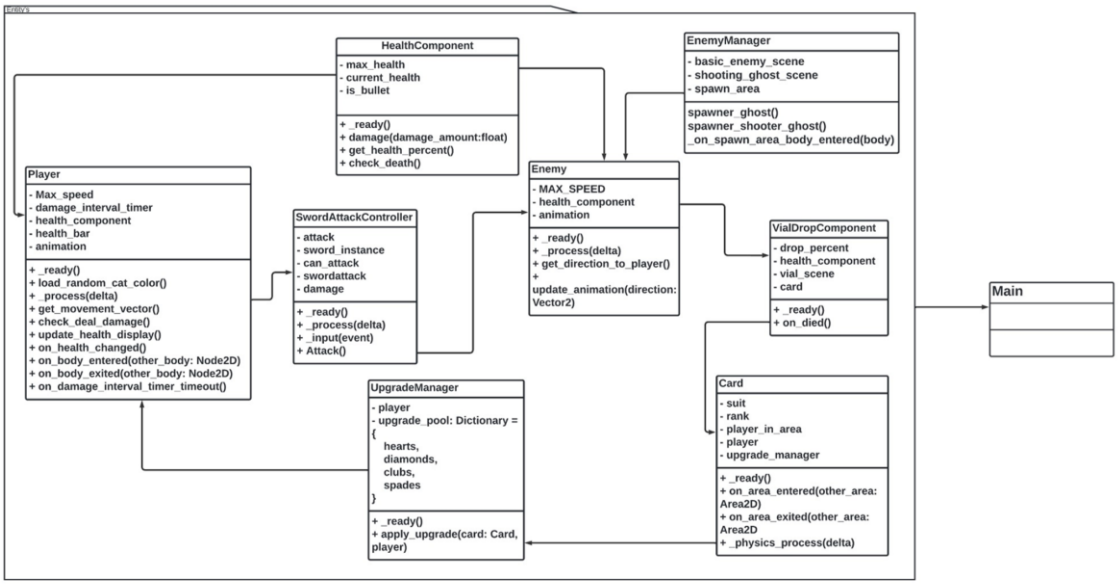

8

# ДІАГРАМА ДІЯЛЬНОСТІ ІГРОВОГО ПРОЦЕСУ



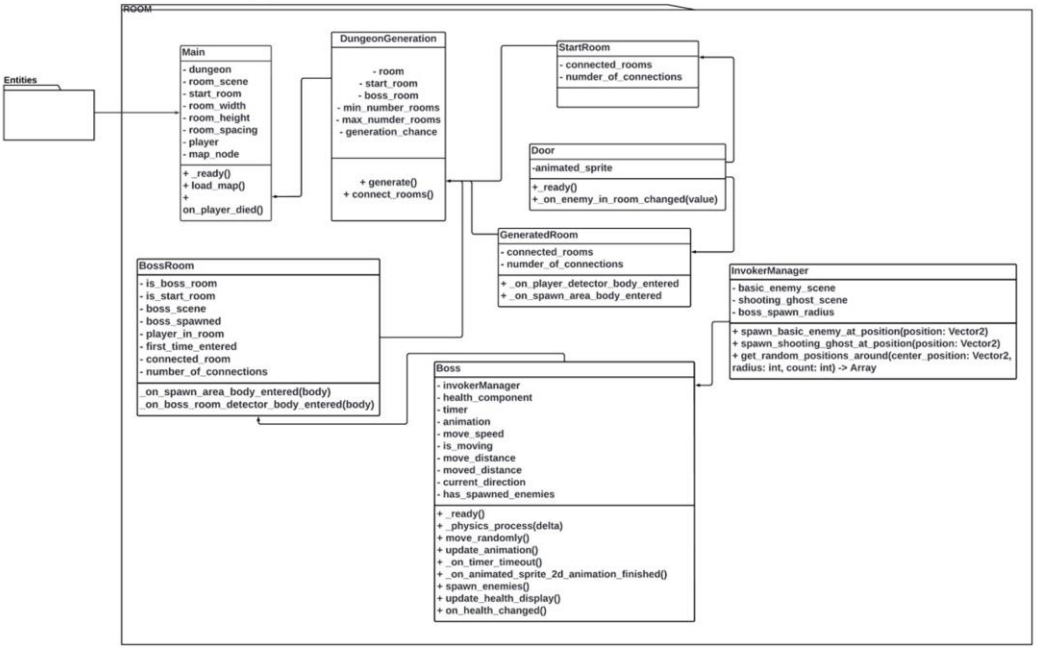

9

# ДІАГРАМА КЛАСІВ СУТНОСТЕЙ



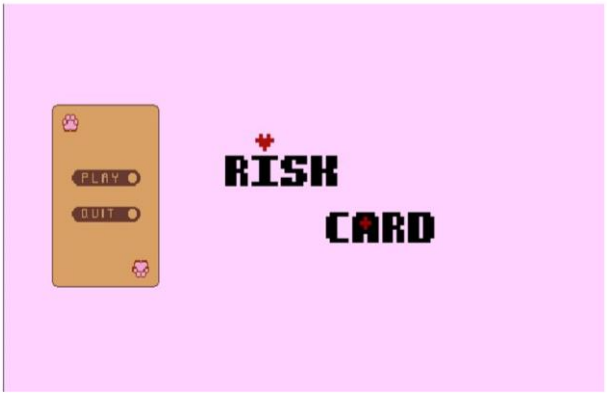

10

## ДІАГРАМА КЛАСІВ РІВНЯ ТА БОСА




11

## ЕКРАННІ ФОРМИ



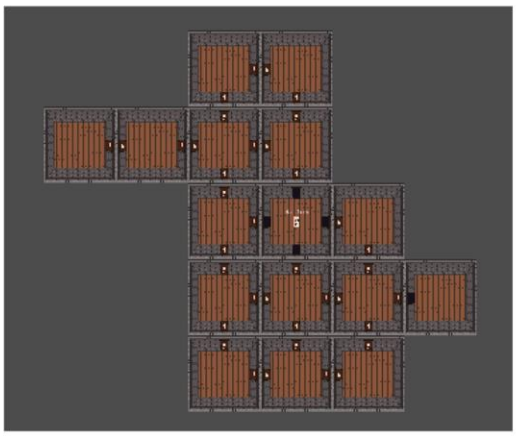

Головне меню


Меню смерті головного героя

12

# ЕКРАННІ ФОРМИ


Меню паузи



Приклад згенерованого рівня

13

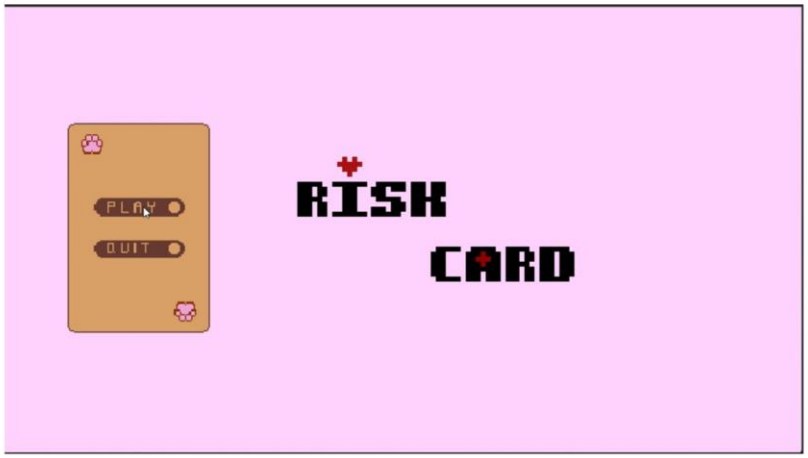
# ЕКРАННІ ФОРМИ


Кімната з босом


Вікно перемоги

14

# ДЕМОНСТРАЦІЯ РОБОТИ ЗАСТОСУНКУ




15

# АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Куріс А.С., Золотухіна О.А. Визначення типових характеристик гри жанру "Roguelike". Матеріали IV Всеукраїнської науково-практичної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. 15.05.2024, ДУІКТ, м.Київ. К.:ДУІКТ. С.164-165.

2. Куріс А.С., Золотухіна О.А. Вплив передових технологійна ігрову індустрію. Матеріали V Всеукраїнської науково-практичної конференції молодих вчених та здобувачів вищої освіти присвяченої Дню науки (17 травня 2024 р.). За ред. Г.В. Жосан, Г.О. Димової та ін. Херсон-Кропивницький: Видавництво ФОП Вишемирський В.С., 2024. С. 81-82.

16

# ВИСНОВКИ

1. Проведено аналіз існуючих ігор в жанрі "Roguelike", що дозволило визначити їхні основні особливості та ігрові механіки

2. Визначеноконцепт гри, особливості геймплеюта вимоги до розробкигри в жанрі “Roguelike”. В якості ключових вимог визначено такі як перманентнасмерть та процедурне генерування рівнів. Для покращення геймплею концепт гри включає появу випадкових карт для посилення сил персонажу

3. Розроблено алгоритми, що забезпечують реалізацію вимог, в тому числі, алгоритми для перманентної смерті та процедурногогенеруваннярівнів.

4. Створено основні 2D об`єкти гри, такі як кімнати, вороги та головний герой з виокристанням програмному забезпеченню “Aseprite”. Основні ігрові механіки, такі як рух героя, підбираннякарт та пересуваннямiж кімнатами, реалізованіза допомогоюігрового рушія Godot та мови програмування GDScript.

5. Розроблено тестові сценарії для тестування гри «Card risk», які включають кейси тестування інтерактивностіта ігрового процесу, тестування графічних компонентівта звукового супроводу. Всі тестові сценаріїзавершеноз позитивнимрезультатом.

# ДОДАТОК Б ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
extends Node
class_name HealthComponent

signal died
signal health_changed

@export var max_health: float = 10
@export var is_bullet = false as bool
@export var is_player = false as bool
@export var is_boss = false as bool
@onready var enemy_manager = load('res://assets/manager/Enemy/enemy_manager.tscn')
var  current_health



func _ready():
	current_health = max_health


func damage(damage_amount:float):
	current_health = max(current_health - damage_amount, 0)
	health_changed.emit()
	Callable(check_death).call_deferred()


func get_health_percent():
	if max_health <= 0:
		return
	return min(current_health / max_health, 1)


func check_death():
	if current_health == 0:
		died.emit()
		owner.queue_free()
		if is_bullet == false:
			Event.enemy_in_room -= 1
		if is_boss == true:
			Event.boss_in_room = false
		if is_player == true:
			Event.enemy_in_room = 0
	if number_colliding_bodies == 0 or not damage_interval_timer.is_stopped():
		return
	health_component.damage(1)
	damage_interval_timer.start()
	print(health_component.current_health)


func update_health_display():
	health_bar.value = health_component.get_health_percent()
	Global.health_bar_value = health_component.get_health_percent()


func on_health_changed():
	update_health_display()


func on_body_entered(other_body: Node2D):
	number_colliding_bodies += 1
	check_deal_damage()


func on_body_exited(other_body: Node2D):
	number_colliding_bodies -= 1


func on_damage_interval_timer_timeout():
	check_deal_damage()


func player_shop_method():
	pass
extends Node
```

```
var room = preload("res://environment/Generate_room/generated_room.tscn")

var start_room = preload('res://environment/Start_room/start_room.tscn')


var min_number_rooms = 7

var max_numder_rooms = 15


var generation_chance = 20



func generate(room_seed):
	seed(room_seed)
	var dungeon = {}
	var size = floor(randf_range(min_number_rooms, max_numder_rooms))
	dungeon[Vector2(0,0)] = room.instantiate()
	size -= 1


	while(size > 0):
		for i in dungeon.keys():
			if randf_range(0, 100) < generation_chance:
				var direction = randf_range(0, 4)
				if(direction < 1):
					var new_room_position = i + Vector2(1, 0)

	if(!dungeon.has(new_room_position)):

	dungeon[new_room_position] = room.instantiate()
					size -= 1

	connect_rooms(dungeon.get(i), dungeon.get(new_room_position), Vector2(1, 0))


				elif(direction < 2):
					var new_room_position = i + Vector2(-1, 0)

	if(!dungeon.has(new_room_position)):

	dungeon[new_room_position] = room.instantiate()
					size -= 1

	connect_rooms(dungeon.get(i), dungeon.get(new_room_position), Vector2(-1, 0))


				elif(direction < 3):
					var new_room_position = i + Vector2(0, 1)

	if(!dungeon.has(new_room_position)):

	dungeon[new_room_position] = room.instantiate()
					size -= 1

	connect_rooms(dungeon.get(i), dungeon.get(new_room_position), Vector2(0, 1))


				elif(direction < 4):
					var new_room_position = i + Vector2(0, -1)

	if(!dungeon.has(new_room_position)):

	dungeon[new_room_position] = room.instantiate()
					size -= 1

	connect_rooms(dungeon.get(i), dungeon.get(new_room_position), Vector2(0, -1))


	return dungeon


func connect_rooms(room1, room2, direction):
	if room1.connected_rooms[direction] == null and room2.connected_rooms[-direction] == null:
```

```
                room1.connected_rooms[direction] = room2
                room2.connected_rooms[-direction] = room1
                #room1.number_of_connections += 1
                #room2.number_of_connections += 1
extends Node



@export var player: PackedScene

@export var upgrade_pool: Dictionary = {
        "hearts": [],
        "diamonds": [],
        "clubs": [],
        "spades": []
}

func _ready():
        for suit in upgrade_pool.keys():
                for rank in ["6", "7", "8", "9", "10", "jack", "queen", "king", "ace"]:
                        var upgrade = CardsUpgrade.new(suit, rank, "Описание улучшения для " + rank)

                        upgrade_pool[suit].append(upgrade)



func apply_upgrade(card: Card, player):
        match card.suit:
                "hearts":
                        match card.rank:
                                "ace":
player.Max_speed *= 2
                                "king":
player.Max_speed *= 1.9
                                "queen":
player.Max_speed *= 1.8
                                "jack":
player.Max_speed *= 1.7
                                "10":
player.Max_speed *= 1.6
                                "9":
player.Max_speed *= 1.5
                                "8":
player.Max_speed *= 1.4
                                "7":
player.Max_speed *= 1.3
                                "6":
player.Max_speed *= 1.1
                "diamonds":
                        match card.rank:
                                "ace":
Global.attack_damage_scale * 2
                                "king":
Global.attack_damage_scale * 1.9
                                "queen":
Global.attack_damage_scale * 1.8
                                "jack":
Global.attack_damage_scale * 1.7
                                "10":
Global.attack_damage_scale * 1.6
                                "9":
Global.attack_damage_scale * 1.5
                                "8":
Global.attack_damage_scale * 1.4
                                "7":
Global.attack_damage_scale * 1.3
                                "6":
Global.attack_damage_scale * 1.2
                "clubs":
                        match card.rank:
                                "ace":

                                        player.health_component.max_health += 10

                                        player.health_component.current_health += 10
                                "king":

                                        player.health_component.max_health += 9

                                        player.health_component.current_health += 9
                                "queen":

                                        player.health_component.max_health += 8
```

```
                player.health_component.current_health += 8
                        "jack":
                player.health_component.max_health += 7
                player.health_component.current_health += 7
                        "10":
                player.health_component.max_health += 6
                player.health_component.current_health += 6
                        "9":
                player.health_component.max_health += 5
                player.health_component.current_health += 5
                        "8":
                player.health_component.max_health += 4
                player.health_component.current_health += 4
                        "7":
                player.health_component.max_health += 3
                player.health_component.current_health += 3
                        "6":
                player.health_component.max_health += 2
                player.health_component.current_health += 2
            "spades":
                match card.rank:
                        "ace":
Global.attack_scale_value *= 2
                        "king":
Global.attack_scale_value *= 1.9
                        "queen":
Global.attack_scale_value *= 1.8
                        "jack":
Global.attack_scale_value *= 1.7
                        "10":
Global.attack_scale_value *= 1.6
                        "9":
Global.attack_scale_value *= 1.5
                        "8":
Global.attack_scale_value *= 1.4
                        "7":
Global.attack_scale_value *= 1.3
                        "6":
Global.attack_scale_value *= 1.1
extends Node2D


class_name Card


@export_enum("hearts", "diamonds", "clubs", "spades")
var suit : String = 'hearts'
@export_enum("6", "7", "8", "9", "10", "jack", "queen",
"king", "ace") var rank : String = 'ace'


var player_in_area : bool = false
var player : CharacterBody2D
@onready var upgrade_manager =
load("res://assets/manager/Upgrade/upgrade_manager.ts
cn").instantiate()


func _ready():
        randomize()  # Инициализация генератора
случайных чисел
        var suits = ["hearts", "diamonds", "clubs",
"spades"]
        var ranks = ["6", "7", "8", "9", "10", "jack",
"queen", "king", "ace"]


        # Выбор случайной масти и ранга
        suit = suits[randi() % suits.size()]
        rank = ranks[randi() % ranks.size()]


        player =
get_tree().get_first_node_in_group("player")



        $Area2D.area_entered.connect(on_area_entered
)
        $Area2D.area_exited.connect(on_area_exited)
```

```
		$Sprite2D.texture = load("res://resources/cards_sprite/" + rank + "of" + suit + ".png")


func on_area_entered(other_area: Area2D):
		player_in_area = true
		if other_area.get_parent() is CharacterBody2D:
			player = other_area.get_parent() as CharacterBody2D


func on_area_exited(other_area: Area2D):
		player_in_area = false
		if other_area.get_parent() is CharacterBody2D:
			player = null


func _physics_process(delta):
		if player_in_area and Input.is_action_just_pressed('use') and player != null:
			queue_free()
			upgrade_manager.apply_upgrade(self, player)
extends CharacterBody2D
const MAX_SPEED = 50

@onready var health_component = $HealthComponent
@onready var animation = $Sprite2D

func _ready():
		pass


func _process(delta):
		var direction = get_direction_to_player()
		velocity = direction * MAX_SPEED
		move_and_slide()
		update_animation(direction)


func get_direction_to_player():
		var player_nodes = get_tree().get_nodes_in_group('player')
		if player_nodes.size() > 0:
			var player_node = player_nodes[0] as Node2D
			return (player_node.global_position - global_position).normalized()
		return Vector2.ZERO



func update_animation(direction: Vector2):
		if direction.x > 0:
			animation.play("right")
		elif direction.x < 0:
			animation.play("left")
		elif direction.y > 0:
			animation.play("down")
		elif direction.y < 0:
			animation.play("up")
```