Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Міроника Миколи Вікторовича
Прізвище, ім'я, по батькові студента(ки)

на здобуття ступеня вищої освіти Бакалавра

Ігровий застосунок у жанрі «rogue-lite» з використанням технології Unity
Назва теми

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.200168.01.16.ПЗ

Виконав студент IV курсу, група ІПЗ-20-1 _____ Микола МІРОНИК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник канд. пед. наук, доцент _____ Наталія ПРАВОРСЬКА
Науковий ступінь, вчене звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер ст. викладач _____ Ганна БЕДРАТЮК
Посада Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

6 червня 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
02 01 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Міроника Микола Вікторовича
Прізвище, ім'я, по батькові студента

1. Тема роботи Ігровий застосунок у жанрі «Rogue-lite» з використанням технології Unity

Керівник роботи Праворська Наталія Іванівна канд. пед. наук, доцент.
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 01.03.2024 р. № 5
2. Строк подання студентом роботи на кафедру 01.06.2024 р.
3. Вихідні дані до роботи Матеріали переддипломної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження предметної області та постановка задачі, проєктування програмного забезпечення, програмна реалізація та тестування, висновки

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Три креслення: *(вказуєте назви всіх трьох креслень*
1. UML діаграма прототипів класів
2. ER діаграма класів системи гравця
3. Діаграма потоків даних

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Бевратюк І. І.		06.06.24
Антиплагіат	Форкун Ю. В., доцент	22.05.24	06.06.24

7. Дата видачі завдання «_02_» __січня_ 2024 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Збір матеріалу за темою кваліфікаційної роботи (КвР); дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02.2024	
3 Проектування програмного забезпечення	21.02 – 20.03.2024	
4 Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2024	
5 Написання вступу, загальних висновків, оформлення джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог стандартів	01.05 – 25.05.2024	
6 Попередній захист КвР	Травень 2024	Згідно графіка
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.05.2024	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2024	

Студент _____ Микола МІРОНИК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Наталія ПРАВОРСЬКА
Підпис Ім'я, ПРІЗВИЩЕ

АНОТАЦІЯ

Тема кваліфікаційної роботи: Ігровий застосунок у жанрі «Rogue-lite» з використанням інструменту Unity

Автор проекту: Міроник Микола Вікторович.

Керівник проекту: Праворська Наталія Іванівна.

Пояснювальна записка: 75 с., 43 рис., 3 дод., 36 джерел.

Графічна частина: 18 слайдів.

ВІДЕОІГРА, ГРА, C#, UNITY, ROGUE-LITE, ROGUE

Метою проекту є розробка унікального жанру відеогри, що представить змогу пограти у відеогру де рівні генеруються автоматично, що дозволить проходити гру не один раз, а також зменшить рівень стресу у гравця.

У кваліфікаційній роботі було проведено дослідження предметної області та постановку задач, спроєктовано архітектуру та структуру системи, розроблено і протестовано застосунок, а після підведено підсумки зробленої роботи.

Для реалізації було використано мова C#, та ігровий двигун Unity.

В результаті було реалізовано відеогру в жанрі «rogue-lite», який надає змогу проходити гру де рівні генеруються випадково, що підтримує різноманіття ігрового процесу під час будь-якого проходження відеогри.

06.06.2024
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.200168.01.16.ПЗ	Пояснювальна записка	75		
2	А4		Завдання на кваліфікаційну роботу	1		
3	А4		Анотація	1		
			Графічні документи			
4	А4		Презентаційні матеріали	18		
5	А3	КвРІПЗ.200168.01.16.ПЗ	Діаграма потоків даних	1		
6	А3	КвРІПЗ.200168.01.16.ПЗ	ER діаграма класів системи гравця	1		
7	А3	КвРІПЗ.200168.01.16.ПЗ	UML діаграма прототипів класів	1		

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ. 200168.01.16.ПЗ			
Виконав		Міроник М.В.		06.06.21	Ігровий застосунок у жанрі «Rogue-lite» з використанням інструменту Unity	Літ.	Арк.	Аркушів
Керівник		ПраворськаН.І.		06.06.21			1	1
Н. Контр.		Бедратюк І.П.		06.06.21		ХНУ, ІПЗ-20-1		
Зав. Каф.		Бедратюк Л.П.		06.06.21	Відомість документів			

ЗМІСТ

ВСТУП

У давнину, коли Інтернет та спорт були лише мрією у віддаленому майбутньому, життя вибудовувалось навколо простих задач і задоволень. Кожен день виглядав одноманітно, а нудьга, як вічний супутник, часом призводила до абсурдних ситуацій.

Наші предки не могли собі уявити життя без викликів і випробувань, які приносило полювання. В теорії це можна назвати першою грою, першим змаганням, перший спортивний інтерес людства. Уміння здобувати добич стало виміром сили, розуму та кмітливості. Хто принесе в печеру більше м’яса та шкури вважався найсильнішим та найкмітливішим, а якщо людина вбила небезпечного хижака і принесла його тіло як трофей то без урочистих почестей та поваги не обійтись.

Сьогодні ми живемо в світі з безліччю розваг та розвинутих технологій, але пам'ятаймо про те, що коріння нашої розваги походять з давнини, з часів, коли кожен день був справжнім випробуванням для виживання і саморозвитку. Коли основною задачею людини було – виживання.

Саме в такі часи почався стрімкий розвиток наукових дисциплін. Вчені, філософи та мислителі стали новими героями, вони відкривали та досліджували нові горизонти знань, які не тільки розширювали розуміння світу, але й служили основою для технологічного та культурного прогресу.

Таким чином, в епоху, коли старі заняття втрачали свою роль, наука та розвиток інтелектуальних дисциплін стали новими джерелами цікавості, розвитку та прогресу для людства.

Вони стали тим самим світилом, завдяки кому людство в двадцять першому столітті має стільки розваг та цікавих для вивчення дисциплін. Завдяки ним людство має пиво, шахи, математику, фізику, біології, тощо.

Саме тоді постало питання в розвагах не стільки фізичних, скільки розумових. Потрібні були заняття, які би не стільки навантажували мозок, а й

давали йому відпочити від повсякденної інтелектуальної діяльності, але так, аби мозок не був «відключений» повністю.

На сьогоднішній день людству відомо, по крайній мірі, дві найдревніших гри, які були популярні здебільшого в інтелектуальних колах або світських кругах тих часів.

Цікавість іграми, такими як «Сенет» та «Ура» (Гра двадцяти квадратів), полягала не лише у їхній загадковій будові і правилах, але й у їхньому сприйнятті, як суперницького протистояння [1].

Завдяки торговим караванам та мандрівникам, ці ігри поширилися по світу, перетворившись на популярні забави не лише серед «вищого» суспільства, а й серед звичайних громадян, які висловили свій інтерес. Таким чином, вони стали не тільки формою розваги, а й засобом спілкування та конкуренції, який сприяв розвитку інтелекту та стратегічного мислення серед звичайного населення.

Після «Сенет» та «Ура» з'явилися інші ігри, які були або модифікаціями вже існуючих, або зовсім оригінальними концепціями, наприклад, «Го»[2].

Це свідчить про постійний розвиток та еволюцію ігрової культури, що відбувалася на протязі віків. Таким чином, ігри стали важливою частиною життя суспільства, відображаючи його цінності, інтереси та культурні особливості в певний період часу.

Цікава особливість даних ігор полягала в тому, що вони потребували двох гравців, які ставали супротивниками на ігровому полі. Це може свідчити або про спортивний інтерес, або про інстинктивне бажання перемогти, притаманне людям. Інстинкт битви, змагання.

Таким чином, навіть у розважальних справах людство завжди знаходило спосіб протистояти собі подібним. Це забезпечило популярність настільних ігор серед різних верств суспільства, від «вищого» до звичайних громадян. Такий інтерес до гри сприяв її поширенню та популяризації, роблячи її важливою частиною культурного життя.

На довгий час ігри зупинили свій розвиток в різноманітті, аж до початку двадцятого століття. В час, коли люди винайшли Інтернет, обчислювальні машини, а також відображення графіки на екранах.

Від першої гри для двох гравців, яка представляла собою теніс, до високобюджетних проектів на сотні людей, де працювали відомі сценаристи, режисери та актори. Сучасні ігри стали тим, що дозволяє людству не тільки відпочивати, а й розвиватись та збагачуватись культурно.

Цей еволюційний шлях довів, що ігри можуть бути більш ніж просто розвагою. Вони стали інструментом для навчання, вираження та спілкування. Їхній вплив на культуру та суспільство стало невід'ємною частиною нашого життя та побуту.

Ігри можуть бути не лише захоплюючою історією або складними ігровими механіками, але й уособленням ідеалу мистецького генія. Завдяки комбінації історії, естетики, музики та ігрового процесу вони привертають мільярди гравців по всьому світу. Сучасні ігри вийшли за межі простої розваги для «гіків» та «ботанів», перетворившись на окремий ринок та важливу складову сучасної культури.

Ігри стали не лише причиною розвитку робототехніки та програмування, але й сприяли загальному технологічному прогресу інформаційних систем. Вони впливають на розвиток таких наук, як психологія, надаючи можливість вивчати поведінку та реакції людей в різноманітних ситуаціях.

Вони стали не лише засобом розваг, але й способом зниження загального рівня стресу в людей. Ігри задовольняють інстинктивні потреби, такі як: полювання, азарт бою, суперництво між гравцями. Це сприяло зменшенню загального рівня агресивності та «дикості» серед людей. Ті, хто грає у відеоігри, стають більш ерудованими та обізнаними, мають більше коло інтересів.

Характерною особливістю цього жанру є те, що гравець має досліджувати безкінечно випадково згенеровані рівні своїм персонажем, обраному перед початком гри, якщо гра надає таку можливість. У цих іграх гравець змушений

пройти через серію рівнів, боротися з ворогами та пошукати скарби, щоб покращити своє спорядження та навички для подальшого прогресу в грі.

Особливістю є те, що при «смерті» персонажа він втрачає все своє спорядження та вміння, і гравцеві доводиться починати зі стартової точки, що додає елемент виклику та напруги.

Актуальність теми кваліфікаційної роботи бакалавра полягає в поєднанні високої реіграбельності, випадково генерованих рівнів та непередбачуваності геймплею, що властиві жанру «Rogue-lite». Це відкриває широкі можливості для дослідження та удосконалення ігрових механік.

Застосування технології ігрового двигуна Unity дозволяє реалізувати концепції та ідеї у стійкому та потужному ігровому середовищі, а ще й зробити це безоплатно. Метою індивідуального завдання є створення відеоігрового застосунку в жанрі «rogue-lite» за допомогою технології Unity.

Для успішної роботи над темою кваліфікаційної роботи бакалавра потрібно дотримуватись наступного:

– дослідити теоретичну частину предметної області;

– провести детальний аналіз здобутої інформації та виділити основні моменти які зможуть допомогти в успішній реалізації теми;

– дослідити ігри та їх будову як окремих програмних застосунків, класифікацію, основні моменти які роблять гру – грою;

– провести аналіз уже існуючих на даний момент варіантів рішень на ринку відеоігор;

– на основі отриманої інформації спроєктувати архітектуру та структуру системи відеогри;

– провести дослідження технології Unity;

– розробити та реалізувати відеоігровий застосунок, який буде задовольняти гравців;

– провести тестування розробленого відеоігрового застосунку та виправити помилки.

1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Змістовний аналіз предметної області, її структурних та функціональних особливостей

Для того, аби почати дослідження та аналіз предметної області потрібно визначити, що таке є «гра» та «розвага». Визначивши дані поняття можна буде набагато успішніше шукати, обробляти та аналізувати отриману інформацію.

Визначення «гри» та «розваги» в контексті відеоігор та програмного забезпечення можна віднести до такої області в створенні ігор як «геймдизайн».

Геймдизайн відповідає за застосування художньої погляду для створення сюжету відеоігри та технічної допомоги в її створенні для перетворення її на готову до гри відеоігру. Крім того, ігри часто створюються для освіти, фізичних вправ або експериментів. Таким чином, це поєднує різні галузі, такі як: комп'ютерні науки та програмування, творче письмо, психологія та візуально-графічний дизайн [6].

Згідно книзі за авторством Джессі Шелла, відомого геймдизайнера та автора відеоігор поняттю «гра» можна надати наступні значення: гра – те, у що ми граємо; іграшка – це те, із чим ми граємо; іграшка – це предмет, з яким граємо; хороша іграшка – це предмет, що приносить фан; фан – це задоволення із сюрпризами [10].

Якщо приймати вищеописаний текст як визначення «гри», нібито стає все зрозуміло, але не повністю. Адже дане визначення можна віднести майже до всього, що є у світі. Саме тому потрібно звернутись до відомих мислителів минулого, психологів та філософів які в свій час намагались дати визначенню куди більш глибокій абстракції – розвага [10].

Фрідрих Шиллер казав: «Гра – це безцільна витрата надлишкової енергії». Що є невірним визначенням [10]. Адже навіть найстародавніша гра «Ура» чи «Сенет» мали просту та зрозумілу ціль – перемогти свого супротивника [11].

Кеті Зален та Ерік Циммерман запропонували свій варіант визначення «гри» у своїй книзі «Rules of Play»: «Гра – це вільний рух усередині закритої структури.» Звучить надто абстрактно та спочатку незрозуміло, але якщо дивитись в корінь визначення та розшифрувати дану загадку то стає цікаво. Вони намагались описати «гру» не стільки, як технічний продукт, а в загальному. До прикладу: гра промінчиків сонця на стіні або дзеркалі. Але це надто обширно, тому є потреба шукати далі [10].

Відомий американських філософі іспанського походження Джордж Сантаяна надав майже ідеальне визначення гри: «Гра – це все, що робиться спонтанно і заради гри.» [10]. Логічно, що інколи ігри бувають спонтанними, люди можуть сидіти і від нудьги захотіти покидатися камінням в воду? Чи це не є грою? Але чи обов'язкова спонтанність для гри? Відповідь: ні. Адже можна, як запланувати гру з кимось чи з собою, так і спонтанно робити щось, що подобається [12].

Саме тут знадобиться минуле визначення де ключової частиною гри є «фан». Процес задоволення, той самий стан коли людина відчуває задоволення, але людина може відчувати задоволення при інших процесах, саме тому був введений термін «фан»[13].

Після того, як було дане визначення грі та розвазі. Потрібно зробити декомпозицію гри з двох сторін: зі сторони гравця та розробника. Адже зрозумівши, що таке відеогра з погляду цих двох сторін можна буде створити відеогру, а не програмне забезпечення яка симулює гру.

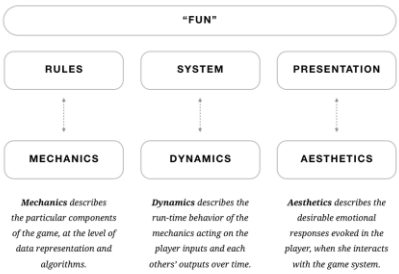


Рисунок 1.1 – Декомпозиційна схема «гри» зі сторони гравця та розробника

В той час, як розробник дивиться на відеоігровий застосунок з лівою сторони, гравець з правої. Перше, що помічає гравець в відеогрі, коли розпочинає грати, входить в головне меню чи дивиться трейлер, тощо. Це естетику, презентацію вигляду відеогри[7].

Музика, графіка, стиль будівель, предметів, спорядження, вигляд не ігрових персонажів, вигляд аватара гравця, голоси і тому подібне. Це все є естетикою відеогри. Перша зона, в яку поринає гравець, саме вона працює, як першочерговий зацікавлювач гравця, той самий гачок, який повинен привернути його уваги. Естетика працює також, як маскування технічних недоліків гри. Гравець може пробачити в грі деякі моменти, якщо картинка перед його очима буде настільки сильно вражати його, що він або не зверне увагу на недоліки або сам собі скаже: «При такому старанні над візуальною частиною прощається»[8].

Надалі, гравець, через призму візуальної сторони відеогри, починає бачити «нутрощі» відеогри. Це те, що дозволяє грі рухатись, це можна назвати «м'язами» гри, якщо приводити аналогію з людиною. Це динаміки та системи. Те, що дозволяє гравцю діяти, його можливості та дії. Системи, та їх угрупування, які називаються – динаміками, дозволять гравцю зробити ту чи іншу дію. Системи – це угрупування механік (про що далі буде написано). Механіки складають системи, системи – складають динаміки. Динаміки – це двигун гри, те, що приводить її до руху та дозволяє гравцю керувати ігровий процесом [13].

Останнім компонентом, який гравець майже не помічає, це – механіки та правила гри. В той час, як система є угрупуванням механік, механіка є окремим об'єктом, який був спроектований з урахування впливу на нього інших механік. І разом вони створюють системи. Це «скелет» гри, те, що визначає її можливості в подальшому, та можливості гравця. Вони і створюють правила, яким слідує гравець, усвідомлено чи ні, але гра завжди веде гравця туди куди їй потрібно згідно установлених правил[14].

Зі сторони гравця нібито все просто: «побачив – порухався – зробив згідно правилам». Нічого складного, гра сама веде гравця навіть, якщо він цього не помічає. В той же час зі сторони розробника все набагато складніше.

Розробка відеогри розпочинається з механік та правил. Вони і створюють «скелет» гри. Механіки взаємодіють між собою, передають дані, інформацію, і в кінці формують правила гри. Розробка та проєктування механік є складною роботою адже потрібно враховувати не тільки, те що хоче геймдизайнер або розробник [15]. А й враховувати дії гравця, що є небезпечною для психологічного здоров'я роботою та марнотратством часу. Але потрібно здогадуватись, що саме потрібно гравцеві, куди він може нажати, що використати. Але ніколи не потрібно намагатись вирахувати, що зробить гравець [16].

Після того, як були розроблені та спроектовані механіки, які і визначили правила гри. Їх потрібно об'єднати в системи, а системи в динаміки. До прикладу: система розвитку героя. В ній є: здоров'я, витривалість, магічна сила, показники сили, мудрості і тому подібне. Але це все маленькі механіки, які і створили систему розвитку героя. Здоров'я, як окремий об'єкт залежить від інших об'єктів, тощо. Механіка мудрості дозволяє гравцеві робити: дія А, …, дія N [17].

За окремістю механіки не можуть працювати поодинці, виключно в зв'язку з іншими механіки.

Після того, як були розроблені системи та динаміки лишається напевно найпростіша та найнепередбачуваніша частина розробки відеогри. Це – візуальний дизайн. Чому найпростіший? Як правило і розробка та ідея відеогри розпочинається з цього, з історії, придумали цікавий дизайн. Протягом всього часу розробки механік, системи та динамік – відділ дизайнерів працював над історією, дизайном локацій та персонажів, які потім експортуються в відеогру. Чому найнепередбачуваніша? Як-не-як – це візуальний стиль, який подобається одній людині та не подобається іншій. Це аксіома: «Якщо щось подобається

одній групі людей – інша група це ненавидить». Саме тому дуже тяжко вгадати, як саме відреагує теоретичний користувач на візуальний стиль [18].

На рис. 1.2 представлено декомпозицію відеогри на основні компоненти.

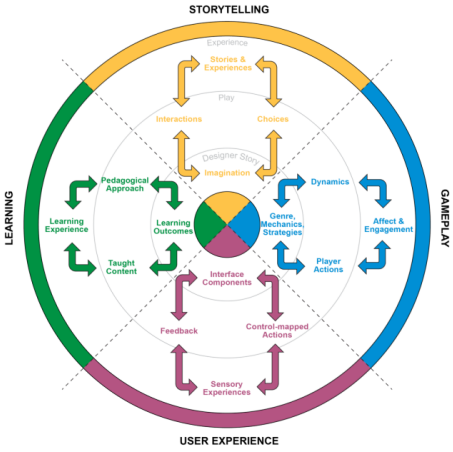


Рисунок 1.2 – Декомпозиція відеогри на основні компоненти

Проаналізувавши декомпозицію відеогри зі сторони гравця та розробника та визначивши, що є визначенням «гри», потрібно провести декомпозицію відеогри, як окремого об’єкту незалежно від гравця, розробника чи інвестора.

Згідно рисунку 1.2 гра має наступні важливі чотири складові:

– оповідання історії;

– геймплей (ігровий процес);

– користувацький досвід;

– вивчення.

Як видно з рисунку, кожна складова складається із трьох шарів: досвід, гра, історія дизайнера. В подальшому буде розглянуто кожен модуль даних складових в контексті того шару, в якому вони знаходяться.

Почнемо зверху та будемо рухатись по часовій стрілці.

Оповідання історії:

– історія та досвід який получає гравець пізнаючи дану історю. Історія може, як містити в собі важкі теми та змусити гравця думати над нею, так і бути простою, але зрозумілою історію, в якій гравець може розслабитись або

віднайти щось нове. Вона знаходиться в зоні «Досвід», що свідчить про те, що взаємодіючи з даним модулем даної складової гравець пізнає новий досвід та отримує враження;

– інтерактивність та вибір, які знаходяться в зоні «Гра». Під час проходження історії у відеогрі, гравець може робити вибір у діалогах та впливати на хід самої історії – це називається інтерактивність. Коли гравець робить ту чи іншу дії, або вибирає певну репліку, він може впливати на хід історії, що робить проходження історії цікавим та хвилюючим моментом;

– уява, те з чого народжується історія, знаходить в зоні «Дизайнерська історія». Все починається з геймдизайнера, він дає ідею, рух, вектор, стиль. А в контексті оповідання історії – це ті самі, не сказані прямо, цікаві факти або приховані секрети світу або історії.

Геймплей (ігровий процес):

– вплив та залучення гравця. В зоні «досвід», який отримує гравець – це є вплив на гру та ігрові системи, шляхом гри в неї та залучення гравця для подальшого дослідження відеогри. Гравець отримує досвід граючи в гру, досліджуючи її, її механіки;

– динаміки та дії гравця. В зоні «гра» знаходяться уже ключові складові гри, які і роблять її грою. Динаміки, які дозволяють гравцю робити ту чи іншу дії, які оформляють кінцеві правила гри. Та дії гравця, весь той набір ігрових можливостей та запропонувань від розробника гравцеві;

– жанр, механіки, стратегії. В зоні досвіду геймдизайнера знаходиться «скелет» гри, який і створює та проектує геймдизайнер.

Користувацький досвід:

– чуттєвий досвід: музика, звуки, візуальна частина гри. Це все є чуттєвим досвідом гравця. Звуки та музика створюють відповідний настрій та атмосферу. Візуальний стиль гри оповідає історію локації, показує гравцеві, що саме це за місце. Це все впливає на гравця на психологічному рівні.

– зворотній зв’язок та контрольно-відображення дій. Гравець повинен розуміти, що він робить. Якщо натиснути кнопку «Рухатись вперед», його

аватар повинен йти вперед. Відкриваючи, наприклад, меню створення предмету гра, повинна візуально та текстово повідомляти гравцеві, в якому саме меню він знаходиться та, що саме від відкрив.

– компоненти інтерфейсу – початок всього і кінець. Те, за допомогою чого гравець взаємодіє з програмним продуктом.

Вивчення:

– досвід навчання. Граючи в гру – гравець вчиться. Він вчиться грати, думати, аналізувати, що є досвідом. Аби перемогти боса потрібно вивчити його. Аби пройти загадку потрібно проаналізувати її. Аби досягти певного результату – потрібно виконати певні осмисленні дії;

– педагогічний підхід та викладання змісту. Серед розробників будь-яких програмних продуктів є аксіома: «Користувач навіть з покроковим керування не зрозуміє і 10% програми». Якщо говорити просто – розумовий рівень середньостатичного користувача не настільки високий аби він міг користуватись будь-яким програмним продуктом без підказок. Особливо це помітно у відеоіграх. Гравець, як дитина, якій потрібно помагати на кожному кроці, давати явні та неявні підказки, хвалити за правильні дії. Гравець потрібен розуміти, що саме відбувається та може статись;

– результати навчання – є тим, що вивчив та зрозумів гравець під час процесу гри.

Після детального аналізу визначення «гри» та структурного аналізу гри, її основних компонентів, які притаманні їй потрібно проаналізувати жанр гри, яка вказана в темі кваліфікаційної роботи бакалавра.

«Rogue-lite» – це субжанр відеоігор, який бере вихідні основи жанру «Rogue-like» і адаптує їх, роблячи ігри менш покаральними та більш доступними для ширшого кола гравців [19]. В той час, як «Rogue-like» ігри характеризуються строгою постійною смертю (permadeath), де гравець мусить починати гру з нуля після кожної смерті, «Rogue-lite» пом'якшує цю вимогу, зберігаючи деякі елементи прогресу між спробами [20]. Це відкриває двері для

більш стратегічного глибокого досвіду, де навіть невдачі ведуть до довготривалого розвитку персонажа та здібностей.

Основні характеристики жанру «rogue-lite» [21]:

– зменшені наслідки смерті: в «Rogue-lite» іграх система постійної смерті втрачає свою безкомпромісність. Наприклад, у грі «Rogue Legacy», гравці передають деякі здобуті ресурси своїм нащадкам, що дозволяє кожному новому персонажу стартувати з певних переваг. Це забезпечує відчуття прогресу та зростання, навіть після численних невдач;

– більша доступність та варіативність у геймплеї: ігри «Rogue-lite» зазвичай пропонують більш різноманітні способи геймплею і можливості адаптації до стилів гри різних користувачів. Це може включати різноманітність вибору класів персонажів, здібностей, а також різні рівні складності. Такі ігри, як «Binding of Isaac» або «Enter the Gungeon», надають гравцям можливість обирати з багатьох унікальних персонажів та зброї, що впливає на стратегію і стиль гри;

– прогресивне покращення: ця особливість є ключовою в «Rogue-lite» іграх, де інвестиції у покращення бази, обладнання або здібностей мають довгострокові наслідки для всіх майбутніх ігрових сесій. Це забезпечує відчуття постійного розвитку, яке мотивує гравців продовжувати грати;

– іноваційність у механіках геймплею: «Rogue-lite» ігри часто впроваджують новаторські ігрові механіки, що змішують елементи з різних жанрів, вносячи свіжість та унікальність у стандартну формулу. Це може включати елементи головоломок, стратегій управління ресурсами або навіть елементи наративного розповідання, які розширюють традиційні межі жанру.

Жанр «rogue-like» зараз перебуває на вершині своєї популярності та розвитку. Протягом останніх років цей жанр значно розширив свої кордони, інтегруючи елементи з інших жанрів і вносячи зміни, які спонукали розробників і гравців активно дискутувати про те, що насправді визначає «рогалик». Відповідно до цього, представлений проект має на меті часткове відновлення оригінальної сутності «Rogue-like» жанру, відкидаючи додаткові механіки,

позичені з інших жанрів, і зосереджуючись на основних, характерних особливостях.

Націлений на створення відеоігрового застосунку, який відображатиме справжній дух «rogue-like», цей проект дозволить гравцям відчути і оцінити жанр у його майже первісному, але сучасно оновленому вигляді. Завдяки цьому, користувачі матимуть унікальну можливість зануритись у світ класичних «рогаликів», де кожна гра передбачає нові випробування.

Розробка також була висвітлена в тезах Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2023» – Міроник М.В., Праворська Н.І. Ігровий застосунок в жанрі «Rogue-lite» з використанням технології Unity. Збірник наукових праць за матеріалами XV Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2023». Хмельницький. 2023. – 197-200 с [22].

1.2. Аналіз наявного програмно-технічного забезпечення предметної області

Ринок «rogue-like» відеоігор має надзвичайно великі розміри та популярність серед відеоігрової аудиторії. Майже кожен рік виходять або середні за розміром проекти від відомих та не дуже студій. Та стабільно раз в пару років якась велика студія випускає свій проект, який стає новим подихом життя для даного жанру [23].

До найвідоміших представників даного жанру можна віднести такі проекти як: «Enter the Gungeon», «Hades», «Binding of Isaac», «Rogue Legacy». Для успішного результату аналізу було обрано чотирьох найкращих представників жанру «rogue-like» ігор.

Відеогра «Enter the Gungeon» яка зображена на рисунку 1.3:

– ігровий двигун: використовує Unity, що дозволяє легко інтегрувати різноманітні ігрові асети та створювати плавну багатоплатформенну підтримку;

– графіка і стиль: піксель-арт стиль з динамічним освітленням та ефектами, що надають грі вінтажного, але виразного вигляду. Інтенсивна анімація та детальне оформлення персонажів та навколишнього середовища створюють унікальну атмосферу;

– геймплейні механіки: ключові елементи включають процедурно генеровані рівні, великий арсенал озброєнь і предметів. Ігра заснована на вмінні швидко реагувати та стратегічно підходити до кожного нового рівня з новими викликами.

Рисунок 1.3 – Ігровий застосунок «Enter the Gungeon»

Відеогра «Binding of Isaac» яка зображена на рисунку 1.4:

– ігровий двигун: початково розроблена на Adobe Flash, але "Rebirth" вже створена на більш сучасному ігровому двигуні, що підтримує глибшу графіку та кращу процедурну генерацію;

– графіка і стиль: темний, мультяшний стиль з біблійними та готичними мотивами. Кожен рівень і монстр унікально оформлені, що підкреслює похмуру тему гри;

– геймплейні механіки: процедурно генеровані рівні, система покращень через випадково знайдені предмети, які драматично змінюють можливості гравця. Гра вимагає від гравця адаптації до постійно змінюваних умов.

Рисунок 1.4 – Ігровий застосунок «Binding of Isaac»

Відеогра «Rogue Legacy» яка зображена на рисунку 1.5:

– ігровий двигун: Unity. Використання цього двигуна спростило розробку і дозволило впроваджувати складні алгоритми для генерації рівнів та наслідування персонажів;

– графіка і стиль: 2D-графіка з високою деталізацією персонажів та середовища. Стиль напівкартуністий, що легко сприймається, але з достатнім рівнем деталей для створення виразного ігрового світу;

– геймплейні механіки: наслідування, де кожен новий персонаж наслідує характеристики та вміння попередника, але з випадковими змінами, що вносять у гру елементи Roguelike. Різноманітні класи та спадкові травми персонажів створюють унікальні ігрові сесії.

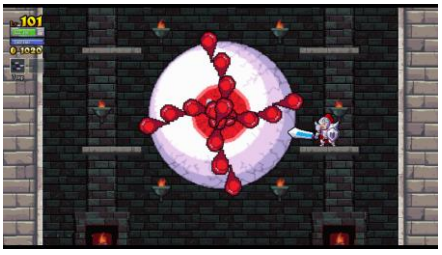


Рисунок 1.5 – Ігровий застосунок «Rogue Legacy»

Відеогра «Hades» яка зображена на рисунку 1.6:

– ігровий двигун: власний двигун розробки Supergiant Games, що оптимізований для підтримки глибокого наративу та складного геймплею;

– графіка і стиль: деталізовані, рукою мальовані спрайти та фони, які візуально розповідають історію. Яскраве, динамічне освітлення та ефекти додають емоційного глибину сценам та битвам;

– геймплейні механіки: глибока наративна складова, поєднана з нескінченними можливостями розвитку персонажа через систему "залежно від смерті" покращень. Динамічний бій з великою кількістю ворогів та босів вимагає від гравця тактичного підходу і стратегічного мислення.

Рисунок 1.6 – Ігровий застосунок «Hades»

Проаналізувавши чотирьох найпопулярніших кандидата на ринку можна прийти до наступного висновку: реалізація на будь-якому складовому рівні гри є дорогим та довготривалим процесом, що в нашому випадку не є актуальним. Саме тому, було обрано піти від зворотнього напрямку. Проаналізуємо найкращого представника «rogue-like» жанру та попробуємо модифікувати його формулу. Тож на основі отриманих даних було знайдено та проаналізовано іншого представника «rogue-lite» жанру. Відеогра «Pixel Dungeon».

«Pixel Dungeon» – це традиційна «рогалик»-гра, яка відома своїм простим інтерфейсом та графікою в стилі піксель-арту. Гру створив незалежний розробник Олег Доля, і вона надихнута попередніми «рогаликами», зокрема «Brogue». Гра кидає виклик гравцям досліджувати рівні підземель, боротися з монстрами та збирати предмети в пошуках «Амулет Йендора». Ігровий процес акцентує на процедурній генерації рівнів, забезпечуючи унікальність кожної гри, і включає постійну смерть, що означає початок з нуля після смерті гравця.

Основні елементи «Pixel Dungeon» включають:

– класи персонажів: гравці можуть вибирати з кількох класів, кожен з унікальними здібностями. Включно з Воїном, Магом, Розбійником і розблокованою Мисливицею, кожен клас пропонує різні стилі гри;

– шгрові механіки: гра є похідною та базується на кахельках, де кожен рух або дія гравця вважається за хід, сприяючи просуванню часу в грі та діям ворогів;

– спорядження та покращення: гравці можуть екіпірувати та покращувати зброю, броню та кільця, що покращують їхні характеристики або надають спеціальні здібності. Однак, деякі предмети можуть бути проклятими, створюючи додаткові виклики;

– споживані предмети: гра включає різноманітні зілля та свитки з ефектами, які спочатку невідомі, поки гравець їх не ідентифікує. Це додає елемент ризику та стратегії до використання ресурсів, знайдених у підземеллі;

– процедурна генерація та бої з босами: кожен рівень підземелля генерується процедурно з боями з босами на певних інтервалах, магазинами та НПС, які з'являються для виконання квестів і торгівлі.

З моменту свого першого випуску, «Pixel Dungeon» яка представлена на рисунку 1.7, стала надзвичайно популярною та впливовою, отримавши високу оцінку за свою здатність балансувати доступність та складність, характерну для «рогаликів». Відкритий код гри сприяв створенню численних модифікацій та спільнотних проектів, що додатково розширили її вплив у геймерській спільноті.

Рисунок 1.7 – Ігровий застосунок «Pixel Dungeon»

Після детального аналізу сучасного трактування жанру «рогаликів», було проведено оцінювання класичного «рогалика» – гри «Pixel Dungeon», яка розроблена з дотриманням основоположних принципів жанру. Однією з

ключових переваг «Pixel Dungeon» є її простота, яка робить гру зрозумілою для гравців; водночас, гра має базовий стиль і геймплей, що зводиться до періодичного натискання кнопок. Ця гра легко освоюється навіть новачками після кількох спроб.

Майбутній проект, над яким ведеться робота, має на меті поєднати простоту «Pixel Dungeon» зі стилем та оформленням «Enter the Gungeon», щоб виділитися серед конкурентів. Бойова система буде запозичена з «Enter the Gungeon» і адаптована, хоча вона не буде настільки динамічною та мальовничою, як у «Hades», через високі вимоги до ресурсів для її реалізації.

1.3. Визначення вимог до програмного забезпечення та технічне завдання

Після глибокого аналізу існуючих рішень у жанрі «rogue-lite», було визначено вимоги до розробки нового відеоігрового застосунку. Він буде представлений у напівкласичному форматі «rogue-lite», з виключенням походової системи для спрощення гри та поліпшення зрозумілості для гравців. Структурна концепція гри детально представлена на рисунку 1.8.

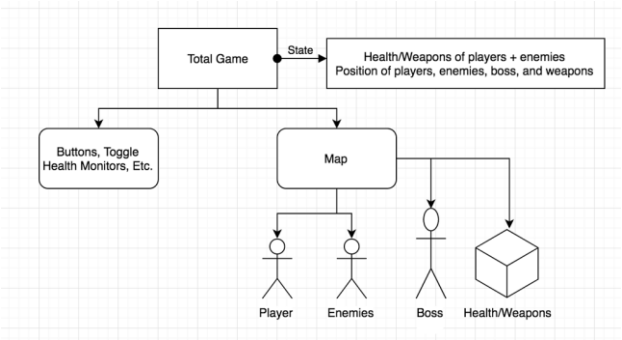


Рисунок 1.8 – Концептуальна структура гри

Вимоги для розробки майбутнього відеоігрового застосунку подаються в напівкласичному стилі жанру «roguelike». Щодо графічного та ігрового оформлення, цей продукт має бути виконаний у піксельній анімації з використанням приємних для ока кольорів, що нетоксичні, для створення

відповідної атмосфери. Інтерфейс гри повинен органічно поєднуватися зі стилем гри, що забезпечує гармонійне сприйняття продукту гравцями.

Проект зосереджуватиметься на розвитку ігрових механік замість сюжетної лінії, що дозволить уникнути затримок у розробці. Бойова система буде адаптована з проекту «Enter the Gungeon» і включатиме механіку «bullet-hell» для динаміки ігрового процесу та введення нових викликів для гравців.

Для підтримки постійного відчуття новизни та авантюризму в грі буде реалізована випадкова генерація підземель. Крім того, гра матиме потенціал для майже нескінченної генерації рівнів, що збільшить її глибину та об'єм. Розробка ведеться на двигуні «Unity» з використанням мови програмування C#, що забезпечить гнучкість та масштабованість проекту для подальших оновлень та розширень без потреби переписування існуючого коду.

Ці вимоги та підходи до розробки забезпечать створення якісного продукту, який буде виділятись на ринку та пропонувати унікальний ігровий досвід своїм користувачам.

Для простоти розуміння майбутнього процесу гри буде представлено схему, яка відобразить ігровий процес майбутнього продукту на рисунку 1.9

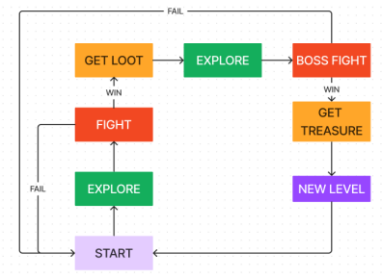


Рисунок 1.9 – «Game flow» проєкту

Отже, підсумовуючи – відеоігровий проєкт повинен мати реалізовані наступні пункти:

– відеогра повинна розроблятись в жанрі «rogue-lite» на базисі відеоігор в жанрі «rogue-like». Це значить, що в грі не буде покрокового ігрового процесу, гра буде динамічною та змушувати гравця приймати рішення швидко;

– бойова система ігрового застосунку повинна підтримувати, як дальній бій, так і ближній бій для гравця та супротивників;

– рівні відеогри повинні представляти собою випадково згенеровані структури, які будуть оформлені за допомогою візуальної графіки, де гравець зможе побачити стиль гри;

– гравець, а саме аватар гравця, повинен мати основні властиві характеристики, які власні подібним аватара;

– спорядження повинно мати основні описові характеристики, які властиві йому в залежності від його типу, класу та рідкості;

– на кожному рівні повинен бути головний супротивник, якого потрібно перемогти аби переміститись на інший рівень відеогри;

– по всьому рівні повинні бути розташовані місця, де гравець зможе отримати нове спорядження для подальшої гри;

– інтерфейс та візуальне оформлення гри повинно бути простим та зрозумілим на інтуїтивному рівні;

– наявність прямої сюжетної лінії чи історії необов’язково.

1.4. Висновки до першого розділу

В першому розділі було проведено глибокий аналіз відеоігор в жанрі «roguelike» та «rogue-lite», досліджено основні характеристики та структуру цих жанрів, а також зібрано важливі визначення гри від відомих авторів та філософів. Основний акцент зроблено на вивченні спонтанності та структурованості ігор, їхньої інтерактивності, а також на важливості дизайнерської уяви у створенні ігрового світу.

З основних визначень ігор виділили визначення, яке описує гру, як «вільний рух усередині закритої структури», що дозволяє глибше зрозуміти фундаментальну природу ігрових механізмів та їхнє значення для гравця. Також, було проаналізовано вплив структурного підходу на розуміння гри, як

інтерактивної діяльності, яка включає: сценарії, ігрові процеси, користувацький досвід та навчання.

Одним із ключових елементів аналізу стало розглядання «Pixel Dungeon», як еталонного «рогалика», який зберігає класичні канони жанру і пропонує простоту та доступність, що робить його зрозумілим широкому колу гравців. На прикладі «Pixel Dungeon» і «Enter the Gungeon» розглянуто можливості поєднання стилістичних та геймплейних елементів для створення унікального ігрового продукту.

Було визначено основі вимоги до майбутнього відеоігрового застосунку, дотримання яких дозволить спроєктувати та реалізувати якісну відеогру незважаючи на відсутність фінансування та невеликі терміни розробки.

У підсумку, цей аналіз вказує на значущість адаптації ігрових механік та дизайну для створення глибоких та захоплюючих ігрових досвідів, які відповідають очікуванням сучасних гравців та виділяються серед конкурентів на ринку.

2. ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Проєктування архітектури та структури системи

На основі даних, які були отримані після дослідження предметної області та аналізу отриманих результатів. Було визначено, що для успішного проєктування архітектури системи потрібно розбити основні системи відеогри на механіки, які і будуть створювати дану систему.

Перед початком розбиття потрібно визначити, які саме системи мають бути реалізовані у відеоігровому застосунку.

Для більш планового визначення основних систем будемо слідувати згідно блок-схеми на рисунку 1.9. До кожного етапу дослідження буде додано відповідну блок-схему, яка дозволить наочно зрозуміти основні компоненти кожного етапу.

Початок гри тобто поява гравця на рівні супроводжується одразу кількома наступними процесами які відбувається майже в один час:

- генерація рівня на якому буде відбуватись ігровий процес;
- виділення окремих територій в яких буде знаходитись спорядження;
- виділення окремої території під головного супротивника;
- ініціалізація появи супротивників в певних кімнатах в певних точках;
- ініціалізація появи гравця на початку рівня;

Беручи до уваги вищеописаний список можемо виділити наступні сутності, які будуть в даному відеоігровому проєкті. Для простоти подальшого згадування буде використано числовий список:

- гравець;
- звичайні супротивники;
- головний супротивник;
- рівень;
- окрема територія під спорядження;
- окрема територія для головного супротивника;

В той час, як сутності один, два та три є унікальними по своєму призначенню модулями, то рівень та окремі території по своєму призначенню та відношенню відносні відеогри можна віднести в окремий клас – генерація рівня.

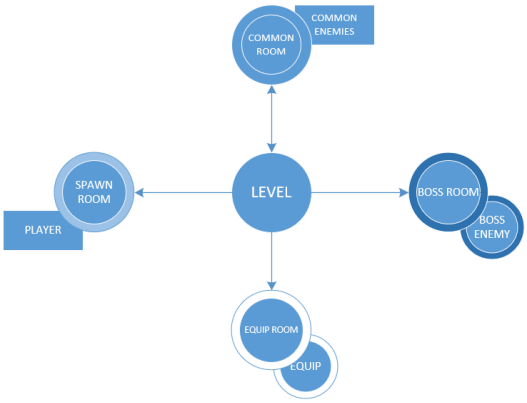


Рисунок 2.1 – Блок-схема взаємозв’язків основних елементів

Перед визначенням основних систем, які належать до даних сутностей потрібно провести їх аналіз. Даний аналіз дозволить наочно нам зрозуміти, за які самі дії та процеси відповідає та чи інша сутність.

Рівень гри, який має в собі території зі спорядження та території головного супротивника можна віднести до однієї сутності – рівень, що зображено на рисунку 2.1:

– генерація рівня;

– виділення зони для головного супротивника;

– виділення зони для спорядження;

– поява гравця в одній із зон;

– поява звичайних ворогів у всіх зонах, окрім зони головного супротивника.

Для кращої наочності була розроблена use-case діаграма (рисунок 1.11). На ньому буде зображено, які саме «дії» має рівень всередині ігрової системи в зв’язку з іншими сутностями.

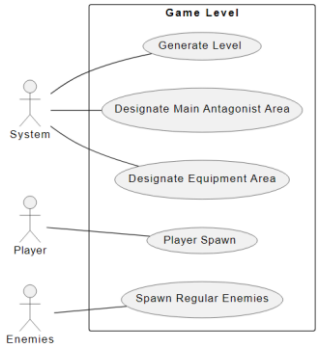


Рисунок 2.2 – Use-case діаграма ігрового рівня

Тепер потрібно визначити механіки які належать до даних систем.

Отже, визначивши основні базиси на яких буде будуватись розроблюваний відеоігровий проєкт, тепер потрібно провести детальний огляд та аналіз кожного базису, визначити його складові та основні компоненти та можливо дані, які потрібні йому для успішного функціонування.

Розпочнемо з рівня, території для головного супротивника та території для спорядження.

Як уже було визначено, всі ці три сутності є однієї загальною сутністю – рівнем. Враховуючи той факт, що розроблювальним відеоігровим застосунком – є гра в жанрі «rogue-lite», а отже рівень повинен генеруватись незалежно від гравця за допомогою певної системи.

Процедурна генерація має багато підходів до генерації будь-чого в сфері інформаційних технологій. Саме тому, використаємо один із методів процедурної генерації, який буде обрано з поміж інших.

На сьогоднішній день на ринку є багато інструментів, які задовольняють потреби незалежних розробників і надають змогу створювати ігри з такими складними системами не створюючи їх самостійно. Але спочатку потрібно визначити, який саме алгоритм процедурної генерації потрібен.

Для процедурної генерації рівнів іграх можуть бути використані численні алгоритми процедурної генерації, кожен з яких має свої унікальні характеристики та переваги. Огляд декількох ключових методів:

– Cellular Automata (Клітинні автомати). Цей алгоритм ідеально підходить для створення більш органічних та натуралістичних середовищ, таких як печери. Він ініціалізує сітку клітин, кожна з яких може перебувати у відкритому або закритому стані. Динаміка клітин змінюється залежно від стану їхніх сусідів за допомогою встановлених правил, що імітує природні формації;

– Random Walk (Випадкове блукання). Підходить для створення лабіринтів і складних коридорів. Алгоритм обирає випадковий напрямок руху з заданої точки, створюючи непередбачувані шляхи, які можуть розгалужуватися або звужуватися, забезпечуючи унікальний ігровий досвід;

– Drunkard’s Walk (П'яний блукання). Варіація методу випадкового блукання, з ще більшою непередбачуваністю. Цей алгоритм ефективний для створення хаотичних і нелінійних структур, де кожна гра відрізняється від попередньої;

– Binary Space Partitioning (BSP). BSP використовується для розділення простору на керовані сегменти, використовуючи дерево розділів. Це забезпечує чітку організацію рівнів з визначеними кімнатами і зонами, підвищуючи якість дизайну рівнів;

– Perlin Noise і Simplex Noise. Ці алгоритми використовуються для створення більш реалістичних і природних текстур чи ландшафтів. Вони ідеально підходять для додавання різноманітності та деталізації у планування кімнат або зон;

– Graph-based Methods (Графові методи). Використання графів для моделювання зв'язків між різними частинами рівня дозволяє створити складні та взаємопов'язані ігрові середовища, де кімнати або зони логічно пов'язані.

Згідно знайдених та проаналізованих алгоритмів та методів для процедурної генерації рівня було обрано графовий метод. Графовий метод – є найкращим варіантом для розроблювальної відеогри. Відеогра повинна сумістити в собі ігровий процес «Pixel Dungeon» та «Enter the Gungeon».

Враховуючи, що на рівні мають бути території, які мають певні характерні особливості, то графовий метод, за допомогою якого можна буде створювати

різноманітні схеми побудови рівня, чітко виділяючи, яка саме частина рівня відповідає за ту чи іншу властивість. Алгоритм роботи даного метода зображено на рисунку 2.3.

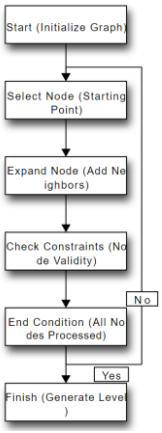


Рисунок 2.3 – Алгоритм роботи графового метода генерації

Графовий метод процедурної генерації рівнів є одним із способів створення різноманітних і непередбачуваних рівнів в іграх, зокрема в жанрі «Rogue-lite». Цей метод використовує графи для представлення рівня, де вузли графа представляють собою кімнати, коридори або інші значимі об'єкти, а ребра – можливі шляхи або двері між ними.

Граф – це математична структура, яка складається з множини вершин (вузлів) та множини ребер, що з'єднують пари вершин. У контексті процедурної генерації рівнів, графи дозволяють моделювати просторові відносини між частинами рівня.

Ключові елементи:

– вершини (вузли): представляють кімнати, майданчики, точки інтересу;

– ребра: представляють коридори або двері, що з'єднують кімнати.

Алгоритм генерації графа:

– ініціалізація: створення початкового вузла графа, який може відповідати входу в рівень або центральній кімнаті;

– розширення: додавання нових вузлів до графа. Це може бути здійснено за допомогою різних алгоритмів, таких як:

1) Random Walk: рандомізоване блукання, де кожен новий вузол додається до випадково обраного існуючого вузла;
2) Binary Space Partitioning (BSP): розділення простору на підпростори, які потім використовуються для генерації кімнат і коридорів;

– оптимізація: після створення базової структури графу, можна використати алгоритми оптимізації для підвищення грайливості. Це може включати забезпечення достатньої зв'язності між кімнатами, видалення непотрібних ребер (що ведуть до надмірної кількості коридорів), або додавання вузлів для підвищення складності.

Отже, як вже і було визначено, в майбутньому буде знайдено інструмент, який дозволить генерувати рівні за допомогою графового метода. Наступною частиною проектування буде опис основних сутностей та їх властивостей.

Першою сутністю буде гравець. Аватару гравця властиві, як правило наступні елементи:

– показник здоров’я;
– показник запасу сил;
– показник магічної сили;
– швидкість гравця;
– інвентар гравця;
– основна «рука» для зброї;
– атака гравця;
– слоти під спорядження;
– слоти під другорядні предмети які можна використати;
– можливість ухилятися від атак супротивника.

Для спрощення подальшої розробки, та оптимізації проєктування для нашої відеогри будуть виділені наступні основні сутності Гравця:

– показник здоров’я;
– час відновлення атаки (cooldown) замість запасу сил;
– р’ять основних слотів під спорядження;

– швидкість гравця;

– можливість ухилятися від атака супротивника (dash);

– час перезарядки ухиляння;

– можливість атакувати.

Для кращої наочності було створено ER-діаграму для подальшої реалізації гравця, рисунок 2.4 та рисунок 2.5 відповідно:

PLAYER		
int	health	Health Points
float	attackCooldown	Time between attacks
string	equipmentSlots[5]	Main equipment slots
float	speed	Movement speed
bool	canDash	Ability to dodge attacks
float	dashCooldown	Cooldown time for dodging
bool	canAttack	Ability to attack

Рисунок 2.4 – Властивості гравця

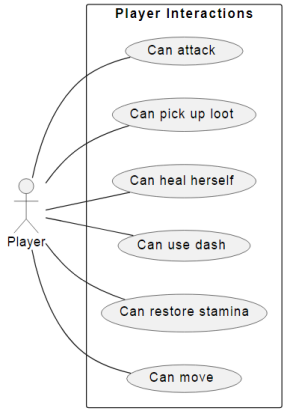


Рисунок 2.5 – Use-case діаграма гравця

Рух, переміщення та атака гравця – є окремим модулем, який можна прив’язати майже до будь-якого об’єкту гри, а саме тому він буде розглянутий окремо від інших.

Наступними для розгляду є супротивники. Для простоти розробки супротивників можна було би використати принцип наслідування створивши батьківській клас, де буде описано основну структуру супротивників і в подальшому просто добавляти нові типи та вигляди супротивників.

Але цей же спосіб не дозволить створювати унікальних ворогів для яких доведеться описувати окремий клас та властивості. Враховуючи, що дана відеогра не має наміру мати великі розміру, а отже для оптимізації розробки буде обрано принцип наслідування. За виключенням головного супротивника, так як його ШІ та здібності будуть кардинально відрізнятись від звичайних супротивників в порівнянні з головним.

Звичайним супротивникам властиві наступні елементи:

– показник здоров’я;

– швидкість ворога;

– його координати на мапі для ШІ;

– статуси на ньому (кровотеча, тощо);

– можливість атаки.

Для наочності буде представлено ER-діаграму властивостей звичайного супротивника, зображено на рисунку 2.6.

ENEMY		
int	health	Health Points
float	speed	Movement speed
string	coordinates	Coordinates on the map for AI
string	statuses	Status effects (e.g., bleeding)
bool	canAttack	Ability to attack

Рисунок 2.6 – ER-діаграма звичайного супротивника

Головний же супротивник буде наслідуваним класом від супротивника, де йому будуть добавлятися уже унікальні властивості. Нижче буде представлено use-case діаграма супротивника, де візуально будуть зображені його дії та можливості в ігровій системі, що зображено на рисунку 2.7.

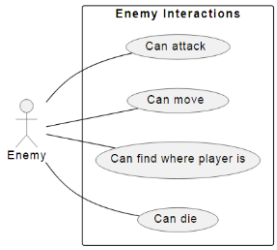


Рисунок 2.7 – Use-case діаграма супротивника

Останньою сутністю, яку потрібно визначити є спорядження. Ми визначили три основні сутності, які будуть присутні в проєкті: рівень, гравець та вороги. Останній, спорядження, є також дуже важливим, але простим елементом відеогри.

Враховуючи невеликий розмір проєкта, то спорядження не матиме найбільш поширених собі властивостей, а будуть представлені звичайні, як для спорядження наступні властивості, для подальшого використання в проєктуванні було створено er-діаграму, що зображено на рисунку 2.8:

– тип спорядження;
– шкода спорядження;
– рідкість спорядження.

EQUIPMENT		
string	type	Type of equipment
int	damage	Damage of equipment
string	rarity	Rarity of equipment

Рисунок 2.8 – ER-діаграма спорядження

Наступні основні елементи гри це є самі методи, які властиві сутностям, такі як: рух, атака, ухиляння від атаки, підбирання предмету, відновлення здоров’я, тощо. Визначивши їх основні властивості та взаємодію з іншими сутностями можна буде приступити до фінального етапу проєктування архітектури та структури, а саме – об’єднання.

Метод руху гравця неможливо застосувати як до супротивників так і до гравця одночасно. Так як саме гравець повинен віддавати команди своєму аватару, а отже, потрібно розробити контролер аватара.

Контролер аватара згідно своїй назві не зможе вміщати в собі тільки рух гравця. Отже, крім руху гравця він буде вміщати в собі наступні можливості системи гравця: рух, атака, ухиляння від атака, піднімання предметів.

Прописувати всі ці можливості в одному класі буде нелогічним та неоптимізованим рішенням. А отже, для проектування контролеру гравця буде використано компонентну архітектуру.

Вона дозволить створити єдиний контролер, який слугуватиме центром управління та буде надавати доступ до відповідних модулів керування гравцем.

Розробка контролера аватара при використанні компонентної архітектури означає, що буде розділино відповідальність на кілька модулів. Це дозволить забезпечити високий рівень гнучкості та зменшити залежності між компонентами:

– MovementController (рисунок 2.9):

1) відповідає за рух гравця;
2) обробляє вхідні команди для переміщення аватара у чотирьох напрямках;
3) може включати логіку для перешкод і фізичних взаємодій;

Рисунок 2.9 – Алгоритм роботи MovementController

– AttackController (рисунок 2.10):

1) управління атаками;
2) включає логіку для різних типів атак (ближній бій, дальній бій);
3) може інтегрувати систему спрямованих атак або прицільного цілiння;
4) в кінці дії завдає шкоду;

Рисунок 2.10 – Алгоритм роботи AttackController

– DodgeController (рисунок 2.11):

1) управління ухиляннням від атак;
2) включає логіку для швидкого переміщення або "доджа" з певним часовим вікном невразливості;
3) запроваджує cooldown між ухиляннями;

Рисунок 2.11 – Алгоритм роботи DodgeController

– ItemInteractionController (рисунок 2.12):

1) обробка логіки підбору предметів;

2) включає в себе детектування предметів у навколишньому середовищі і можливість їх збору;
3) управління інвентарем, додавання або видалення предметів;

Рисунок 2.12 – Алгоритм роботи ItemInteractionController

– Головний контролер (PlayerController) (рисунок 2.13):
1) є центральним вузлом, що інтегрує всі вищезазначені компоненти;
2) обробляє вхідні команди і розподіляє їх між відповідними контролерами;
3) забезпечує єдину точку доступу до функціональностей аватара гравця, знижуючи тим самим зв'язаність;

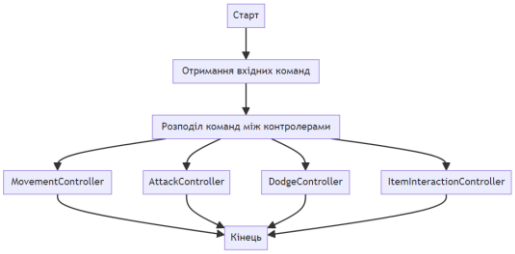


Рисунок 2.13 – Блок-схема взаємозв’язків контролерів

Після того, як було визначено основні контролери, класи та методи за допомогою UML діаграми, буде продемонстровано взаємозв’язок даних сутностей між собою. Дана діаграма допоможе подивитись на структуру системи комплексно (рисунок 2.14).

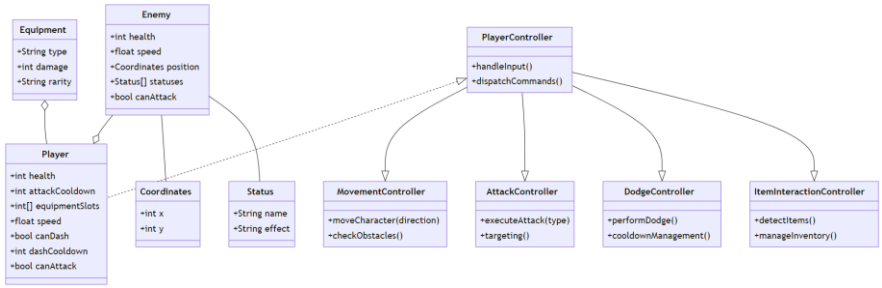


Рисунок 2.14 – UML діаграма прототипів класів

Тепер, коли ми визначили основні складові відеоігрової системи, як вони взаємодіють між один одним та якими властивостями вони наділені, потрібно створити діаграми станів, потоків даних, а також функціональні діаграми, які наочно продемонструють детально, як саме взаємодіють класи з контролерами, які дані передаються і кому.

Даний процес дозволить в майбутньому опиратися на блок-схеми та дані, які вони представляють, що дозволить успішно реалізувати систему.

Розпочнемо з ролі «Гравець». Як вже відомо аватар гравця взаємодія з кількома контролерами, передає в них дані для ініціалізації роботи даних контролерів, та приводить їх в дію. На рисунку 2.15 зображено діаграму станів системи керування гравцем.

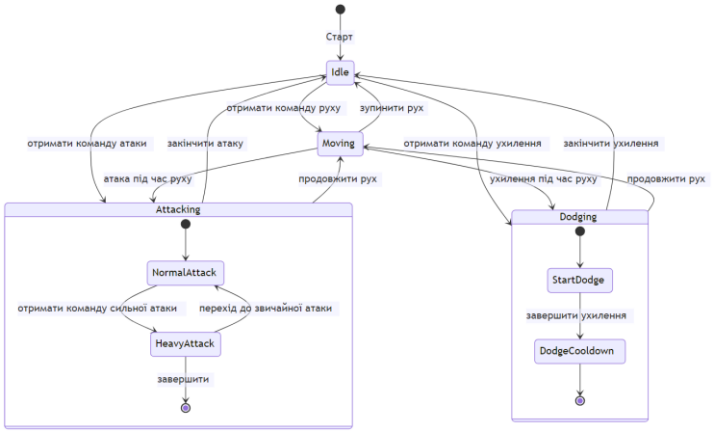


Рисунок 2.15 – Діаграма-станів системи аватара гравця

На даному рисунку наочно відображено, що будь-яка дія гравця відбувається з стану «idle». Даний стан характеризується тим, що аватар гравця

стоїть на одному місті та ним ніхто не керує, але будучи не активним він знаходиться в стані, коли в будь-який момент гравець може ним користуватись.

Як можна помітити на діаграмі-станів присутні блоки, які об’єднують інші стани. Причиною цьому є те, що даний стан має розбіжність у виборі дії або складається із кількох поступових етапів.

Після того, як буде виділено стани які притаманні кожній відеоігровій системі, а також після розробки діаграми прототипів класів даних систем – буде описано потоки даних в межах кожної системи.

Наступним етапом буде створення діаграми потоків даних та ER діаграми для системи аватара гравця, надалі після закінчення роботи над системою гравця будуть розроблені системи супротивників та спорядження. Що дозволить наочно оглянути кожен компонент відеогри (рисунок 2.16).

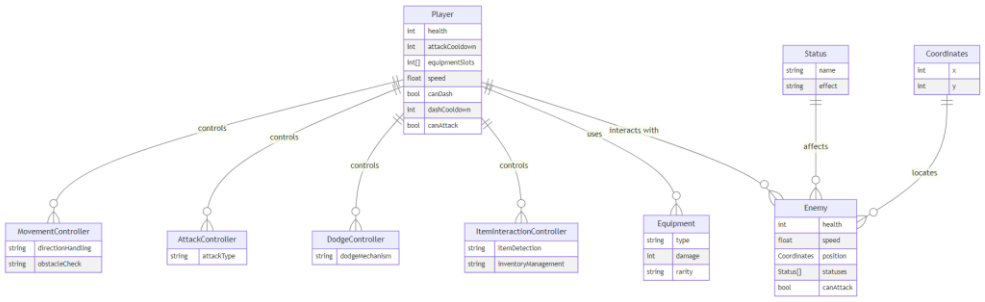


Рисунок 2.16 – ER діаграма класів системи гравця

Система гравця є найбільшою механічною частиною відеогри. Аватар гравця має багато можливостей та здібностей, що мимоволі створює необхідність розробляти велику кількість функцій.

Як уже видно з вищеописаних діаграм, сутність «ворог» не є окремою сутністю та зв’язана з гравцем, але якщо окремо описувати її то виходять наступні діаграми (рисунок 2.17).

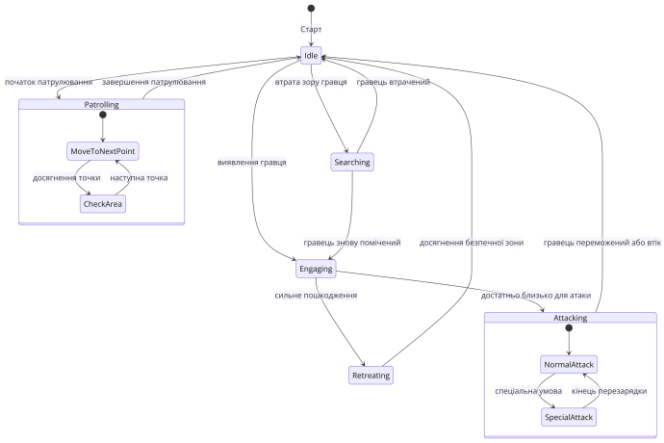


Рисунок 2.17 – Діаграма стані системи супротивників

Аналогічно до системи взаємодії з аватаром гравця – робота даної системи розпочинається з «idle». Причиною тому є те, що при появі на рівні супротивник не шукає гравця постійно, а стоїть на місці очікуючи його поки він ввійде в зону його дії.

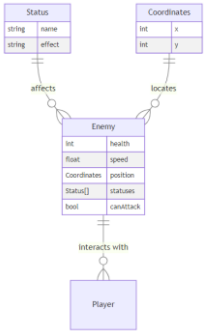


Рисунок 2.18 – ER діаграма класів системи супротивника

До діаграми станів прикладається ER діаграма класів, на основі якої в подальшому буде розроблено діаграму потоків даних. Також на основі даної діаграми можна буде наочно оглянути майбутній вигляд функціоналу системи ворогів у відеогрі. Система класів супротивника є простою, та невибагливою, що продемонстровано на рисунку 2.18.

Не закінчуючи тему системи ворога, потрібно розробити систему штучного інтелекту. Так як вороги повинен сам проводити акти агресії на

гравця, то йому, як об’єкту, який не керується гравцем потрібно закласти алгоритми роботи.

Штучний інтелект ворога повинен вміти наступне (рисунок 2.19):

– при появі ворога в певному радіусі почати атакувати його;

– в залежності від типу атаки (дальній або ближній) наближатись до гравця або тримати відстань та атакувати;

– переслідувати гравця, якщо той спробує втекти;

– якщо гравець втече надто далеко, то ворог повинен «забути» про гравця та вернутись на своє початкове місце.

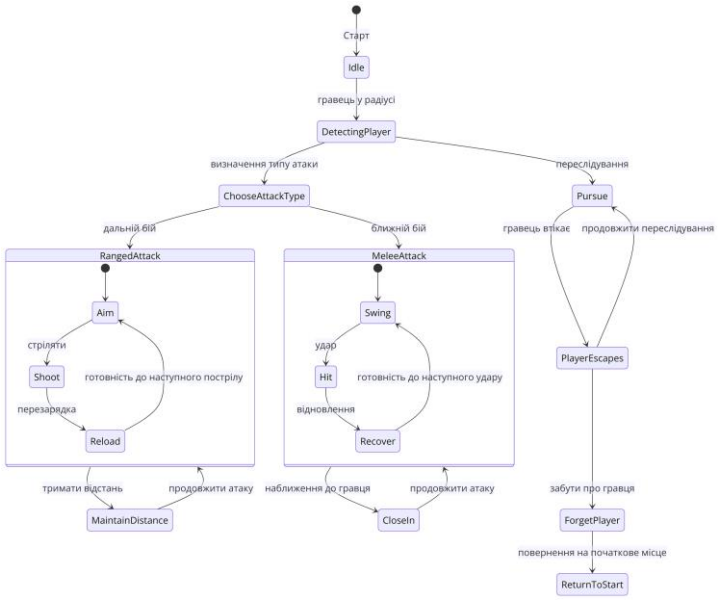


Рисунок 2.19 – Діаграма станів штучного інтелекту ворога

Останніми системами на розгляд є спорядження. В загальному в порівнянні з іншими системами воно є простим за структурою, адже спорядження не прив’язане до функції атаки. Тип спорядження лише має різні способи атаки, які задаються тим чи іншим скриптом.

Якщо прописати спорядженню, що воно має випускати з себе шар, який при контакті з ворогом нанесе шкоду – це буде дальній бій. В той же час, прописавши йому, що при натисканні кнопки атаки в певному радіусі перед

гравцем ворог отримає шкоду – це ближній бій. В загальному змінюється лише вигляд предмета, яким володіє гравець.

Останнім етапом проєктування архітектури та структури системи відеогри – визначення даних, які будуть передаватись через систему. Для наочності дані будуть відображені на діаграмі потоків даних, рисунок 2.20.

Так як система буде працювати на основі скриптів та подій, то більша частина передачі даних це команди від натискання клавіш на клавіатурі. Від клавіші залежить, який саме контролер буде використано, яку саме команду виконає другорядний контролер.

Згідно схеми для деяких подій дані повинні надійти з інших контролерів. Такі події як «ухиляння» та «атака» залежать від їх паралельних подій.

Ухиляння від атаки залежить від того, яка саме клавіша руху була активна в той момент часу коли гравець натиснув на клавішу «ухильнутись».

Атака зброєю залежить від того яка саме зброя знаходиться у гравця в інвентарі, а саме тип зброї.

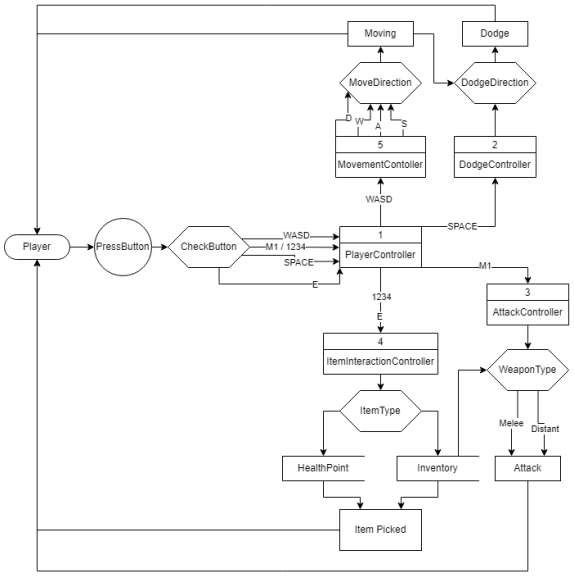


Рисунок 2.20 – Діаграма потоків даних

2.2. Проєктування інтерфейсу користувача

Інтерфейс користувача є важливою складовою будь-якого програмного забезпечення. Завдяки йому користувач отримує візуальну інформацію під час роботи з програмним забезпеченням, що дуже важливо для ефективного використання програмного забезпечення.

Дана ж аксіома працює і для відеоігор. Особливо для відеоігор. Гравець під час процесу гри отримує всю важливу для гри інформацію через графічний інтерфейс відеогри.

За допомогою інтерфейсу розробники не тільки доносять інформацію до гравця, а й можуть оповідати історію та сюжет відеогри. Стиль та спосіб взаємодії з грою також залежить від інтерфейсу.

До прикладу можна привести інтерфейс такої серії відеоігор як «Fallout», що продемонстровано на рисунку 2.21.

Рисунок 2.21 – Приклад оформлення інтерфейсу серії відеоігор «Fallot»

Згідно сюжеті серії «Fallout», даний пристрій, який є на руці головного героя, називається «Pip-Boy». Даний пристрій використовувався, як зараз смартфон або планшет, персональна переносна електронно-обчислювальна машина. Через даний пристрій в серії ігор гравець може оглядати свій інвентар, стан здоров’я, мапу світу, тощо.

Про що і було вище сказано, інтерфейс є важливим елементом будь-чого з чим може взаємодіяти людина.

Для того аби спроєктувати гарний дизайн графічного інтерфейсу, потрібно виділити основні складові гарного дизайну. Як правило, виділяють вісім основних складових гарного дизайну користувацького інтерфейсу:

– чіткість;
– лаконічність;
– знайомість;
– відгук;
– послідовність;
– привабливість;
– ефективність;

Для нашої відеогри буде розроблено простий, але зрозумілий інтерфейс. Він буде мати в собі всю важливу для гравця інформацію, не бути перенавантаженим лишніми даними, а надавати гравцю інформацію, яку він матиме змогу швидко запам’ятати та використовувати не дивлячись куди саме він натискає.

Інтерфейс повинен бути схожим на інтерфейс інших «rogue-like» відеоігор аби гравець міг інтуїтивно користуватись ним. Він не повинен бути перенавантаженим великою кількістю інформації, а забезпечити гравця необхідними для гри даними.

Важливою властивість є «відгук». Натискаючи на певний елемент інтерфейсу, або зробивши щось нове у відеогрі – гравець повинен отримувати інформацію візуально або звуком. Це потрібно для того аби гравець розумів, чи дали результат його дії чи ні.

Послідовність стоїть поряд із знайомістю. Використання одних і тих самих елементів інтерфейсу, які і будують його дозволить гравцеві швидко опанувати новий графічний інтерфейс та вміло використовувати його інтуїтивно розуміючи який елемент і за, що він відповідає.

Привабливість інтерфейсу забезпечує приємне використання користувачем його ж. Візуальне та графічне оформлення інтерфейсу здебільшого залежить від багатьох факторів:

– наявність фінансів для створення свого оформлення;

– вміння створити своє оформлення;

– наявність готового оформлення.

Визначивши основні складові якісного графічного користувацького інтерфейсу, тепер потрібно визначити основі складові графічного інтерфейсу нашої відеогри.

До них можна віднести наступні елементи:

– головне меню;

– інтерфейс під час гри;

– інвентар.

Три основні складові користувацького які будуть надалі спроєктовані та зображені на відповідних рисунках 2.22 та 2.23.

Рисунок 2.22 – Головне меню

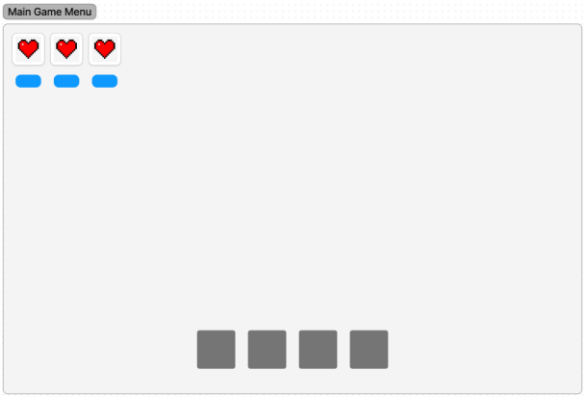


Рисунок 2.23 – Меню під час гри

Під час проєктування інвентаря було помічено, що гравцеві не надається настільки великий по розмірах інвентар аби робити для нього окреме частину. Саме тому, інвентар стане частиною «меню під час гри» (виділено сірим).

Після проєктування «mockup» дизайну графічного інтерфейсу, потрібно визначитись із початковим дизайном та графічним оформленням інтерфейсу. Після чого виділити основні елементи інтерфейсу, які будуть характеризувати той чи інший аспект відеогри.

Палітра оформлення інтерфейсу не повинна бути токсичною та надто яскравою, надто яскраві кольори втомлюють очі та викликають напругу у користувачів, що в подальшому приведе до негативного досвіду у користуванні.

Базові елементи інтерфейсу повинні бути знайомі користувачам. Якщо замість умовних сердець використовувати зірки для позначення кількості здоров'я у аватара гравця, він нічого не зрозуміє.

2.3. Аналіз та вибір технологій і методів реалізації

Адекватний вибір технологій та методів реалізації є важливим аспектом успішності будь-якого програмного проєкту. У сфері розробки програмного забезпечення від правильного вибору інструментів значною мірою залежить, ефективність використання часу та ресурсів команди, а також, в якому саме вигляді та стані перебуватиме продукт в момент випуску [24].

Особливо важливим є цей вибір у контексті розробки ігрових застосунків, де важливу роль відіграють такі фактори, як: продуктивність, візуальна привабливість та зручність використання.

Для системної реалізації всього проєкту буде використано ігровий двигун «Unity Engine». Даний двигун був створений, як безоплатний та доступний всім аналог інших дорогих та вимогливих ігрових двигунів. Він підтримує створення відеоігор майже всіх жанрів та специфікацій.

З поміж інших своїх конкурентів, «Unity» має ряд вагомих переваг [25]:

– мультиплатформність: однією з найбільших переваг «Unity» є можливість створення ігор для багатьох платформ, включаючи «Windows», «macOS», «Android», «iOS», «PlayStation», «Xbox», та багато інших. Це дозволяє розробникам охопити широку аудиторію без додаткових витрат на портування відеоігор;

– інтуїтивний інтерфейс: «Unity» має зручний і зрозумілий інтерфейс, що дозволяє швидко освоїти основні функції і приступити до розробки. Навіть новачки у програмуванні можуть швидко створювати прототипи ігор та експериментувати з різними ідеями;

– широкий набір інструментів та бібліотек: «Unity» надає потужні вбудовані інструменти для роботи з графікою, анімацією, фізикою, звуком, тощо. Це надає змогу розробникам створювати високоякісні ігри без необхідності використовувати сторонні бібліотеки;

– спільнота та підтримка: «Unity» має велику спільноту розробників, що активно ділиться знаннями та ресурсами. Офіційна документація, форуми, навчальні курси та керівництва значно полегшують процес навчання та розробки;

– «Unity Asset Store»: магазин активів «Unity» пропонує тисячі готових моделей, текстур, звуків, скриптів та інших ресурсів, які можна використовувати у своїх проєктах. Це дозволяє значно прискорити розробку та знизити витрати.

Після визначення за допомогою якого саме інструменту буде створено відеогру, необхідно визначити із мовою програмування, на якій буде написано кодову частину проєкту.

Прекрасний вибором буде мова програмування C#. Це пов’язано із багатьма причинами. Основною з яких є – нативність даної мови програмування до ігрового двигуна «Unity». Нативна мова програмування даного середовища розробки стане міцним фундаментом у створенні відеогри.

До основних переваг даної мови відносять наступне [26]:

– синтаксична зручність: C# має зрозумілий та легкий для вивчення синтаксис, що робить її популярною серед розробників. Це дозволяє швидко писати та читати код, зменшуючи кількість помилок;

– об'єктно-орієнтоване програмування: C# підтримує потужні можливості об'єктно-орієнтованого програмування, що дозволяє створювати гнучкі та легко підтримувані додатки. Підтримка таких концепцій, як спадкування, інкапсуляція та поліморфізм, робить код більш структурованим і зрозумілим;

– інтеграція з .NET: C# тісно інтегрована з платформою .NET, що надає доступ до великої кількості бібліотек та інструментів для розробки. Це значно спрощує процес створення складних додатків;

– безпека типів: C# забезпечує строгий контроль типів, що допомагає уникнути багатьох помилок під час компіляції та виконання коду. Це робить програмне забезпечення більш надійним та безпечним.

Під час визначення життєвого циклу розробки, за допомогою якого буде реалізовано гру. Вибір пав на інкрементну модель життєвого циклу створення програмних продуктів. Ця модель характеризується тим, що розробка відбувається поетапно, як і впровадження функціональних компонентів системи. Такий підхід дозволяє розробникам розділити проєкт на менші, більш керовані сегменти, що називаються інкрементами. Кожен інкремент додає нову частину функціоналу до системи, що в підсумку з кожним наступним інкрементом система стає більш повноцінною (рисунок 2.24) [27].

Рисунок 2.24 – Наочне зображення інкрементної моделі

2.4. Висновки до другого розділу

У даному розділі була виконана комплексна підготовка до створення ігрового застосунку. На початкових етапах було спроектовано архітектуру та структуру системи, що включало написання псевдокоду для майбутніх класів та створення відповідних схем, таких як «use-case» діаграми, діаграми алгоритмів роботи та E-R-діаграми. Ці підготовчі заходи дозволили чітко визначити основні компоненти системи та їх взаємодію між собою.

Далі було розроблено «mockup» дизайн інтерфейсу користувача, де були виділені ключові складові якісного дизайну. В результаті було сформульовано основні принципи для майбутнього графічного інтерфейсу: уникати використання яскравих та токсичних кольорів, використовувати знайомі елементи інтерфейсу та уникати перевантаження непотрібною інформацією. Це забезпечить зручність і привабливість інтерфейсу для користувачів.

Завершальним етапом став аналіз інструментів для розробки гри. Було обрано «Unity», як основну платформу, враховуючи її потужні можливості та підтримку мови програмування C#. Також було вирішено застосовувати інкрементну модель життєвого циклу розробки, яка дозволяє поетапно додавати нову функціональність, забезпечуючи гнучкість та можливість своєчасного виявлення і виправлення помилок.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1. Програмна реалізація механік та систем відеогри

З попередніх пунктів відомо про основні та другорядні модулі даної відеогри. Згідно них реалізація буде відбуватись у два етапи: кодова частина та інтеграція в ігровий двигун «Unity», що буде супроводжуватись відповідними рисунками на яких буде зображено кодову частину, та «скелет» даного модуля в «Unity Engine» для кращої наочності.

Першою системою зі своїми механіками для реалізації, буде система «Player». Але, як вже відомо з розділу про проєктування, дана система має не тільки ігрові механіки для реалізації, як наприклад рух чи атака. Даній системі, як і іншим системам, притаманні такі властивості як: графіка та анімація. Враховуючи той факт, що відеогра розроблюється в простому піксельному стилі, а отже графіка є складовою частиною анімації, в той час, як анімація пов’язана з графікою.

Це пов’язано з тим, що піксельна анімація є покадровою анімацією. Наприклад, анімаційні мультфільми та фільми. Це саме відноситься і до піксельної графіки. В зв’язку з цим, будь яку піксельну графіку, яку можна би було використати для реалізації графічної частини у відеогрі потрібно перевіряти на наявність розкадровки. У випадку якщо її немає, задля економії часу потрібно буде шукати іншу графіку, так, як створювати свою покадрову анімацію є довготривалим процесом (рисунок 3.1).

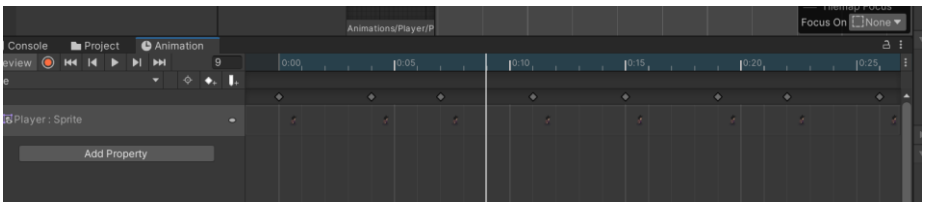


Рисунок 3.1 – Приклад анімування на основі піксельної графіки

Як уже і було написано першою системою для огляду «Player». Дана система характеризується тим, що в ньому було використано «Input Action System». Це умовно нова система, яка отримала велике оновлення та стала заміною застарілої системи ручного кодування управління.

«Input Action System» робить код введення більш чистим і зрозумілим, дозволяє легко підтримувати різні пристрої вводу і спрощує обробку складних схем управління (рисунок 3.2) [28].

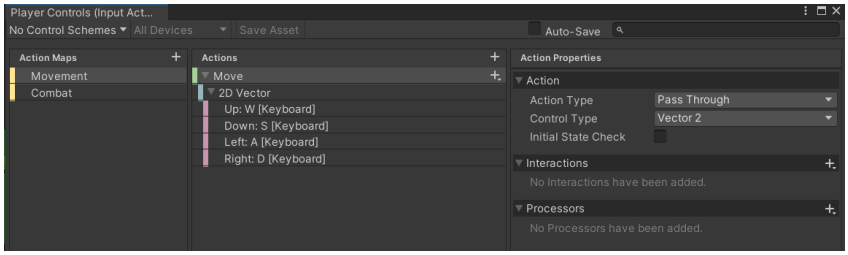


Рисунок 3.2 – Вікно «Input Action System»

Вона дозволяє запрограмувати гравця робити різноманітні дії не заглиблюючись в кодову частину. Дана система автоматично згенерує необхідний клас який не матиме лишнього коду. Також дана система дозволяє реалізувати керування на різноманітних пристроях, таких як: консолі та геймпади якими вони керуються.

Після того, як було створено окрему функцію керування, так як вона на даний момент не прив’язана до об’єкту «гравець». Було реалізовано «PlayerController». Кодову частину якого наведено нижче:

```
public class PlayerController : MonoBehaviour
{
    public bool FacingLeft { get { return facingLeft; } }
    public static PlayerController Instance;
    [SerializeField] private float moveSpeed = 1f;
    [SerializeField] private float dashSpeed = 4f;
    [SerializeField] private TrailRenderer myTrailRenderer;
    private PlayerControls playerControls;
    private Vector2 movement;
    private Rigidbody2D rb;
```

```
    private Animator myAnimator;
    private SpriteRenderer mySpriteRender;
    private float startingMoveSpeed;
    private bool facingLeft = false;
    private bool isDashing = false;

    private void Awake()
    {
        Instance = this;
        playerControls = new PlayerControls();
        rb = GetComponent<Rigidbody2D>();
        myAnimator = GetComponent<Animator>();
        mySpriteRender = GetComponent<SpriteRenderer>();
    }

    private void Start()
    {
        playerControls.Combat.Dash.performed += _ => Dash();
        startingMoveSpeed = moveSpeed;
    }
```

Як уже видно з кодової частини до даного класу підключено наступні системи: «SpriteRenderer», «Animator» та підключено автоматично згенерований клас по управлінню персонажем.

Для забезпечення роботи анімацій, у «Unity» є такий компонент, як «Animation Controller» та «Animation Clip».

«Animation Controller» в «Unity» – компонент, який керує анімаціями персонажів або об'єктів грі. Він визначає, які анімації повинні відтворюватися в ситуаціях, таких як: ходьба, біг, стрибки атака. Компонент дозволяє налаштовувати переходи анімаціями на основі умов подій, наприклад, натискання кнопок або зміни швидкості. Він забезпечує зручний для створення складних поведінок, що дозволяє керувати рухами діями персонажів грі.

«Animation Clip» відповідає за зберігання даних про анімацію, такі як положення, обертання і масштаб об'єктів у певні моменти часу. Він визначає, як об'єкт буде рухатися, змінювати форму або виглядати протягом анімаційного циклу. Використовується в «Animation Controller» [29].

«SpriteRenderer» та «Animator», взаємопов’язані між собою та відповідають за коректну роботу графіки яка підключена до об’єкту («SpriteRenderer»), а також за анімацію при переміщенні аватара гравця («Animator»). Як вже і було вище описано для роботи анімацій використовуються два компоненти. Те, як вони взаємодіють між один одним та передають даних продемонстровано на рисунку (3.3).

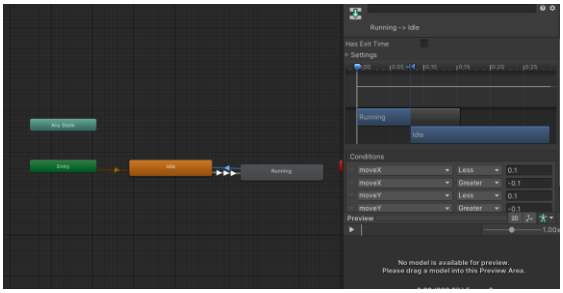


Рисунок 3.3 – Механіка анімації аватара гравця

На рисунку 3.3 зображено роботу анімацій гравця. Від блоку «Idle» який автоматично активує після старту роботи програми анімацію бездіяльності персонажа (він рухається коли стоїть на місці), відразу після отримання команди, які відображаються у вигляді «moveX» та «moveY», що зроблено заради відокремлення механіки анімації задля її стабільності та кращого читання. Програється анімація руху гравця.

Для анімування гравця було використано дві анімації: «idle», «move-left». Це зроблено для оптимізації розміру та швидкої відеогри. Для того, щоб зробити анімацію руху «right», в кодовій частині було реалізовано функцію, яка керується за допомогою курсору миші.

Кодова частина, яка представлена нижче демонструє просту але ефективну функцію. Він змінює напрямок погляду графіки в залежності від положення курсору миші на екрані. Проста перевірка, яка перевіряє чи курсор миші знаходиться лівіше чи правіше осі X відносно екрану.

Після чого в залежності від показнику за допомогою функції «flipX», графіку віддзеркалюють на іншу сторону, якщо ж курсора миші знаходиться на

тій стороні де графіка дивиться в свою звичайну сторону нічого не змінюється. Кодову частину наведено нижче:

```
private void AdjustPlayerFacingDirection()
{
    Vector3 mousePos = Input.mousePosition;
    Vector3 PlayerScreenPoint = Camera.main.WorldToScreenPoint(transform.position);
    if (mousePos.x < PlayerScreenPoint.x)
    {
        mySpriteRender.flipX = true;
        facingLeft = true;
    } else
    {
        mySpriteRender.flipX = false;
        facingLeft = false;
    }
}
```

Кодову частину де було підключено керування гравцем наведено на рисунку нижче:

```
private void PlayerInput()
{
    movement = playerControls.Movement.Move.ReadValue<Vector2>();
    myAnimator.SetFloat("moveX", movement.x);
    myAnimator.SetFloat("moveY", movement.y);
}
```

Дана функція відповідає за приймання команд від автоматично згенерованого класу та зчитує значення вектору руху. Після чого дані передаються в аніматор для передачі даних, щодо руху гравця аби в подальшому активувати анімацію та передати дані для віддзеркалювання графіки руху.

Наступним ключовим елементом системи «PlayerController» є можливість ухилятися від атак супротивників. Цей код додає можливість гравцеві виконувати ривок («Dash»). Під час ривка швидкість руху гравця збільшується, а також активується візуальний ефект. Ривок триває короткий час, після чого швидкість повертається до початкової, а візуальний ефект вимикається. Після

ривка є короткий час очікування, перш ніж гравець зможе виконати ривок знову. Це реалізовано за допомогою наступної кодової частини:

```
private void Dash()
{
    if (!isDashing)
    {
        isDashing = true;
        moveSpeed *= dashSpeed;
        myTrailRenderer.emitting = true;
        StartCoroutine(EndDashRoutine());
    }
}

private IEnumerator EndDashRoutine()
{
    float dashTime = .2f;
    float dashCD = .25f;
    yield return new WaitForSeconds(dashTime);
    moveSpeed = startingMoveSpeed;
    myTrailRenderer.emitting = false;
    yield return new WaitForSeconds(dashCD);
    isDashing = false;
}
```

Після того, як було розроблено кодову частину «dash» аватару гравця, потрібно підключити анімацію цього ж ухиляння. Серед варіантів реалізації анімації ухиляння було обрано не редагувати саму модель персонажа, не робити анімацію перекату, тощо. А додати візуальний ефект швидкості який буде створювати вигляд нібито персонаж на своїх ногах швидко перемістився. Ідея була взяти з кінофільмів.

Дана анімація була реалізована за допомогою «TrailRenderer». «TrailRenderer» – це компонент, який використовується для створення слідів, що залишаються позаду рухомих об'єктів. Він часто використовується для додавання візуальних ефектів, таких як сліди від частинок, сліди від куль, ефекти магії або просто для надання більшої динаміки руху об'єктів (рисунок 3.4).

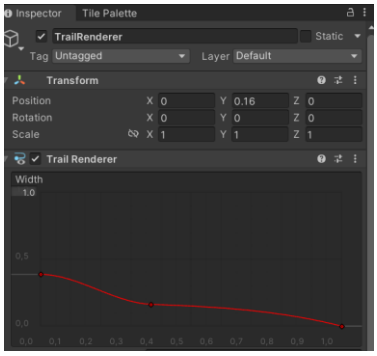


Рисунок 3.4 – Анімація «Dash» аватару гравця, затухання сліду

Анімація ухиляння відбувається за допомогою простої білої лінії, яку згодом було перефарбовано в градієнт. Від чорного до трохи коричневого, так як графіка вигляду гравця здебільшого має коричневий колір. Після того, як активується анімація дана лінія з'являється на точці де був персонаж та розтягується до точки, де персонаж опиниться, створюючи ефект швидкості. Представлено на рисунку 3.5.

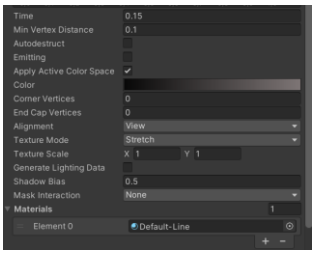


Рисунок 3.5 – Анімація «Dash», налаштування сліду

Наступною системою для огляду буде представлено зброю, алгоритм роботи зброї для всіх видів однаковий.

Для наочності роботи зброї було представлено на рисунку 3.6 нижче.

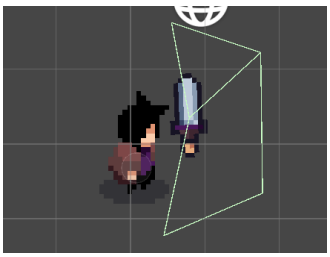


Рисунок 3.6 – Принцип роботи зброї

Атака відбувається не за допомогою об’єкта, як це реалізується в 3D відеоіграх. Для персонажа створюється поле атаки, яке активується в момент отримання команди від натискання кнопки (зелений багатогранник). Як видно на рисунку меч, який тримає аватар гравця менший за область, це зв’язано з тим, що для меча створюється анімація руху (потік вітру від атаки).

Коли гравець натискає кнопку атаки, перевіряється, чи персонаж вже атакує. Якщо ні, атака запускається: активується анімація атаки, вмикається зона для визначення попадання по ворогу, створюється візуальний ефект атаки на екрані. Потім запускається процес, який контролює час між атаками, щоб персонаж не міг атакувати без перерв.

Наступною механікою, яка зв’язана із зброєю є «відкидування». Для того аби візуально повідомляти гравцеві, що атака попала по супротивнику потрібно використати не тільки анімацію отримання шкоди, а й продемонструвати, що зброя має вагу. Також відкидування має глибоку геймплейну важливість, так як здоров’я аватару гравця обмежено, то гравцеві потрібно мати засіб за допомогою якого він зможе захистити себе від напливу великої кількості супротивників. Саме для цього і була реалізована механіка «відкидування». В загальному, алгоритм її роботи дуже простий.

При взаємодії з об’єктом «супротивник», відбувається спочатку отримання шкоди, а потім об’єкт відкидається в протилежному напрямку. Для кращого розуміння принципу роботи кодову частину наведено нижче:

```
public bool GettingKnockedBack { get; private set; }
 [SerializeField] private float knockBackTime = .2f;
 private Rigidbody2D rb;
 private void Awake()
 {
    rb = GetComponent<Rigidbody2D>();
 }
 public void GetKnockedBack(Transform damageSource, float knockBackThrust)
 {
    GettingKnockedBack = true;
```

```
        Vector2 difference = (transform.position - damageSource.position).normalized * knockBackThrust * rb.mass;
        rb.AddForce(difference, ForceMode2D.Impulse);
        StartCoroutine(KnockRoutine());
    }
    private IEnumerator KnockRoutine()
    {
        yield return new WaitForSeconds(knockBackTime);
        rb.velocity = Vector2.zero;
        GettingKnockedBack = false;
    }
```

Цей код реалізує механіку відштовхування персонажа при отриманні удару. Після удару персонаж відштовхується у напрямку, протилежному до джерела удару, із силою, що залежить від його маси. Відштовхування триває короткий час, після чого персонаж зупиняється. Під час цього проміжку він не може рухатися або виконувати інші дії.

Наступною системою для огляду буде «Супротивник». Першою зв’язаною механікою з даним модулем буде саме здоров’я супротивника. В даному класі визначається скільки здоров’я надано супротивника та прописано умови отримання шкоди від атак аватару гравця.

Також в даному класі реалізовано функцію перевірки чи об’єкт «вмер», що зв’язано із здоров’ям цього ж об’єкту. Дана система є уніфікованою та може задіятись для надання здоров’ю будь-якому об’єкту, що дозволить створювати в майбутньому по бажанню різноманітних супротивників.

Даний клас контролює кількість «здоров’я» супротивника. Коли він отримує удар, його «здоров'я» зменшується на певну величину, в залежності від того скільки шкоди наносить зброя. Якщо «здоров'я» досягає нуля, ворог «помирає», перестаючи бути активним об’єктом на мапі.

Одночасно з цим, ворог відштовхується назад від місця удару. Активується поява анімації, яка візуально інформує про отримання супротивником шкоди.

Після чого запускається перевірка стану ворога з коротким очікуванням. Перевіряється, чи ворог «помер», і якщо так, виконується дія, відтворюється ефект смерті з анімацією чи іншими візуальними ефектами.

Наступною системою для розгляду буде – генерація рівня.

У даній відеогрі було використано графовий генератор рівнів, за авторством «Ondřej Nepožitek». Посилання на його GitHub, представлено в переліку джерел [30].

Для того аби вміло та ефективно ним користуватись потрібно знати за якими саме принципами працює даний генератор, які вхідні та вихідні дані він отримує. Для цього було досліджено представлену автором документацію та інструкцію. Після її дослідження та аналізу дана система була зміненою для того аби приймати вхідні дані та об’єкти (гравець, супротивники, тощо).

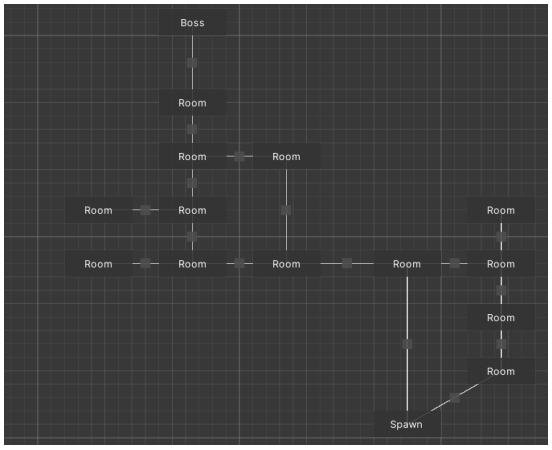


Рисунок 3.7 – Графовий метод генерації рівня

Вся магія генерації рівня за графовим методом в тому, що рівень генерується згідно по взаємозв’язкам між кімнатами. Розміщення, положення завжди буде різне, завдяки чому гравцю не вийде запам’ятати або зрозуміти спосіб генерації рівня (рисунок 3.7).

Основні налаштування відбуваються за допомогою включень уже готових кімнат в кімнат, які повинні використовуватись для генерації рівня. Як видно на рисунку 3.8 є два поля для заповнення.

Верхнє поле відповідає за кімнати, які будуть використані в генерації рівня. Нижнє поле відповідає за коридори та зв’язки між даними кімнатами. Важливим нагадування є те, що, якщо коридори будуть надто короткими, то є великий шанс накладання згенерованих кімнат одну на одну. Саме тому потрібно робити коридори середнього або довгого розміру.

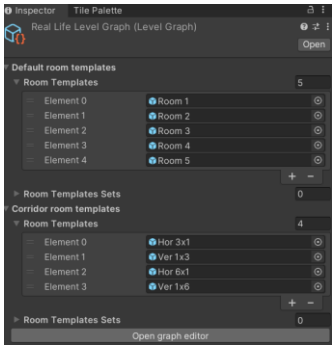


Рисунок 3.8 – Налаштування графового методу

Для кращої наочності на рисунку 3.9 буде продемонстровано одну з кімнат, яка була створена з базової графіки яка є в «Unity». На даному рисунку червоним кольором виділено точки від яких, і до, яких під’єднуються коридори, що дозволяє створювати рівень.

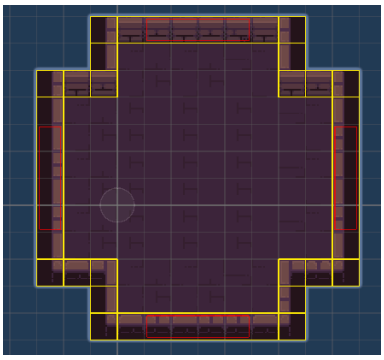


Рисунок 3.9 – Приклад кімнати

Останнім етапом у реалізації відеогри є інтеграція всіх розроблених модулів. В «Unity» інтеграція відбувається за допомогою накидання скрипта на об’єкт та введенні потрібних даних, якщо скрипт передбачає це. Для прикладу

буде представлено аватар гравця. На даному об’єкті висять наступні скрипти та компоненти: «SpriteRenderer» – графіка та вигляд аватару гравця, «PlayerController» – скрипт, який дозволяє гравцеві рухатись та діяти, «Rigidbody 2D» – компонент, який дозволяє взаємодіяти з даним об’єктом, як з 2D об’єктом, використовується в класах для ідентифікації аватару, «Animator» – для анімування гравця, «Capsule Collider 2D» – компонент, який дозволяє гравцеві отримати «реальність», за допомогою нього він не проходить крізь інші об’єкти (рисунок 3.10).

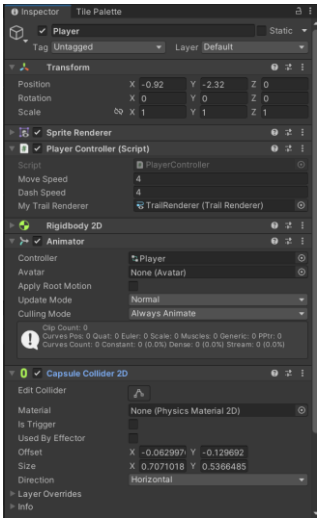


Рисунок 3.10 – Компоненти та скрипти, які були задіяні для роботи аватару гравця

3.2. Вибір та обґрунтування методів тестування додатку

Даний розділ є не менш важливішим аніж інші розділи. У ньому буде обґрунтовано вибір методу тестування додатку. Важливо пам’ятати, що тестування є критично важливим для забезпечення якості та надійності програмного забезпечення, що в свою чергу впливає на задоволення користувачів та успіх продукту [31].

Аби позбавити сумнівів будь-кого, хто забажає оскаржити вищеописане твердження, приведемо приклад з реального життя: зовсім недавно, у 2020 році,

польська студія по розробці відеоігор «CD Projekt RED» випустила гру, над якою вони працювали п'ять років – «Cyberpunk 2077». Основною особливістю гри на старті була неймовірно велика кількість різноманітних багів, деякі з яких були веселими та смішними для гравців, але більшість з них не викликали позитивних емоцій. Погана оптимізація, періодичні краші гри з подальшим викидом гравця на робочий стіл, тощо. Протягом недовгого розслідування був виявлений корінь всіх проблем – халтурне тестування відеогри. Тестування було доручено аутсорсинговій компанії, яка тестувала гру легковажно та надсилала виключно позитивний зворотній зв'язок розробникам, що в кінці вилилося у великий скандал під час старту гри [32].

Отже, підсумовуючи вищеописану інформацію – тестування є вкрай важливим аспектом у розробці, на всіх етапах.

Основною особливість розроблювального програмного застосунку є те, що – це відеогра. Відеоігри з поміж всіх можливих програм характеризуються однією важливою особливістю – інтерактивність. Інтерактивність – це те, що робить гру грою, і те, що є коренем всіх можливих проблем. Надаючи гравцю можливість діяти в межах світу, робити певні дії, взаємодіяти з різноманітними об’єктами – розробники підписують «контракт з дияволом» [33]. Не існує в світі кращих тестерів, руйнівників програм та систем, ніж гравці. По своїй природі – це найбільш хаотичні, неслухняні та руйнівні користувачі систем. Саме тому, підхід до тестування повинен імітувати взаємодію гравця із нею [34].

Тому основою для подальшого тестування стане динамічне тестування в поєднанні з модульним тестуванням, де в подальшому буде проведено інтеграційне тестування системи та механік які їм притаманні.

Даних підхід забезпечить тестування усіх можливих механік та систем під час розробки та по закінченню розробки. Це дозволить розробникам та залученим до діла людям ефективно знаходити баги та проблеми у механіках.

За типом підходу до тестування, вибір є очевидним. Ручне тестування без використання сценаріїв та автоматизованих систем. Нам важливо повністю

імітувати поведінку гравця, саме тому ми маємо діяти як гравець: йти не по сценарії чи правилах [35].

В результаті чого ми матимемо змогу протестувати гру, як на стрес та надійність, адже результат хаотичних дій ніколи невідомий. Даний підхід дозволить в майбутньому створити міцний та витривалий продукт, який не зламається під натиском допитливих гравців [36].

Після визначення з тим, як саме буде тестуватись функціонал відеогри потрібно визначити, які додаткові аспекти притаманні даній відеогрі. Це потрібно для того, аби визначити чи потрібні такі специфічні тестування, як наприклад, тестування безпеки.

Розроблений відеоігровий застосунок немає доступу до мережі та не потребує доступу до мережі інтернет для ефективної роботи, отже тестування безпеки не є важливим.

Також гра не взаємодіє з іншим програмним забезпеченням при роботі, отже тестування на сумісність з різним програмним забезпеченням, яке може взаємодіяти з відеогрою не є доцільним.

Важливим етапом буде тестування сумісності відеогри з операційною системою «Windows», так як випуск гри на консолях або на пристроях від фірми «Apple», потребує їх пристроїв. Саме тому, гра буде доступна виключно на системі «Windows».

3.3. Тестування додатка та аналіз результатів тестування

Тестування потрібно провести над наступними механіками та системами:

– переміщення гравця:
    1) «Dash»;
    2) переміщення по горизонталі та вертикалі;
– атака гравця;
– отримання шкоди супротивником;

– відкидання супротивника після отримання шкоди;

– «Смерть супротивника»;

– генерація рівня;

– неможливість ходити скрізь об’єкти аватару гравця;

Кожен із даних пунктів буде тестуватись власноруч. До кожного із даних систем та механік буде підібрано власний спосіб тестування.

Тестування переміщення гравця включає перевірку плавності та точності руху гравця:

– Dash: перевірка, чи гравець може виконати ривок, та чи правильно анімація і швидкість відповідають очікуванням. Тестуватиметься на різних поверхнях та у різних напрямках;

– переміщення по горизонталі та вертикалі: оцінка реагування на вхідні команди для руху вліво, вправо, вгору та вниз. Перевірка відсутності затримок, збоїв та плавності переходів між напрямками.

Тестування атаки включає перевірку реєстрації атак, їх анімацій та впливу на супротивників.

– перевірка точності та ефективності ударів;

– оцінка візуальних ефектів і анімацій;

– тестування механізму завдання шкоди супротивникам.

Тест отримання шкоди супротивником перевіряє, як правильно супротивники отримують шкоду від атак гравця.

– перевірка коректного зменшення здоров’я супротивника після атаки;

– оцінка візуальних ефектів і реакції супротивника на отримання шкоди.

Тест на відкидання перевірить, чи правильно відкидається супротивник після отримання удару.

– перевірка напрямку та сили відкидання;

– оцінка взаємодії з іншими об’єктами під час відкидання (зіткнення, зупинка).

Тест на «смерть» об’єкта перевіряє коректність механіки смерті супротивника.

– перевірка анімації та ефектів смерті;

– оцінка видалення супротивника зі сцени або його переходу в стан неактивності;

– тестування умов, за яких супротивник вважається мертвим (здоров’я = 0).

Тестування генерації рівня включає перевірку механізму генерації рівнів для забезпечення їх правильності та різноманіття.

– оцінка випадковості та різноманітності згенерованих рівнів;

– перевірка зв’язності кімнат і коридорів;

– тестування наявності всіх необхідних елементів на рівні (перешкоди, вороги, предмети).

Цей тест перевіряє, чи правильно працює колізія між гравцем та іншими об'єктами.

Перевірка, чи гравець не може проходити крізь стіни, перешкоди та інших персонажів.

Оцінка взаємодії гравця з різними типами об'єктів.

Всі тести були успішно пройдені, під час ручного тестування не було виявлено критичних багів, які будуть перешкоджати користувачам грати у відеогру. Швидкодія відеогри на відмінному рівні, під час процесу гри не використовуються надлишкові ресурси персонального комп’ютера.

Тестування пройдено успішно. Тестова авдиторія підтвердила стабільність роботи відеогри.

3.4. Керівництво користувача

Спочатку необхідно скачати інсталяційний файл «game_setup.exe» з наданого посилання або носія, наприклад, USB-диску. Перейдіть до папки, де зберегли файл «game_setup.exe», і двічі натисніть на нього, щоб розпочати

процес інсталяції. Після запуску файлу відкриється майстер інсталяції. Натисніть кнопку «Next» (Далі), щоб продовжити.

Наступним кроком виберіть директорію для встановлення гри. За замовчуванням гра буде інстальована у «C:\Program Files\GameName». Якщо ви хочете змінити місце інсталяції, натисніть «Browse» (Огляд) і виберіть іншу директорію. Після вибору місця натисніть «Next» (Далі), щоб продовжити.

Натисніть кнопку «Install» (Встановити), щоб розпочати процес інсталяції. Дочекайтеся завершення процесу, це може зайняти кілька хвилин. Після завершення інсталяції з'явиться повідомлення про успішне встановлення. Натисніть кнопку «Finish» (Завершити).

Під час завершення інсталяції виберіть опцію «Create a desktop shortcut» (Створити ярлик на робочому столі), якщо така опція доступна. Якщо ви не створили ярлик під час інсталяції, ви можете створити його вручну. Перейдіть до директорії, де була інстальована гра (наприклад, «C:\Program Files\GameName»), знайдіть файл «GameName.exe», натисніть правою кнопкою миші на нього і виберіть «Send to» -> «Desktop (create shortcut)».

Щоб запустити гру, двічі натисніть на ярлик «GameName» на робочому столі. Ви також можете запустити гру з меню «Пуск». Натисніть кнопку «Пуск» (Start), перейдіть до «All Programs» (Усі програми) або «All Apps» (Усі додатки), знайдіть директорію з назвою вашої гри та натисніть на «GameName», щоб запустити відеогру на вашому ПК.

Для керування вашим персонажем у грі використовуйте наступні клавіші:

– W, A, S, D: використовуйте ці клавіші для переміщення персонажа. Натискайте W для руху вгору, A для руху вліво, S для руху вниз і D для руху вправо;

– Space: натискайте цю клавішу для ухиляння. Це допоможе уникати атак ворогів та швидко пересуватися на короткі відстані;

– E: використовуйте цю клавішу для підняття предметів. Підійдіть до предмета і натисніть E, щоб його підняти;

– 1, 2, 3, 4: натискайте ці клавіші для вибору зброї. Кожна цифра відповідає певному типу зброї, яку можна швидко перемкнути під час гри;

– M1 (ліва кнопка миші): натискайте цю кнопку для атаки. Використовуйте її, щоб завдавати ударів ворогам.

Якщо слідувати даній інструкцію можна буде уміло керувати аватаром гравця та гарно провести свій час у даній відеогрі.

3.5. Вимоги до технічних та програмних засобів

Мінімальні вимоги:

– операційна система: Windows 7, Windows 10 (64-біт);

– процесор: Intel Core i3-2100 або еквівалентний;

– оперативна пам'ять: 4 ГБ;

– відеокарта: NVIDIA GeForce GTX 550 Ti або еквівалентна з підтримкою DirectX 11;

– місце на диску: 500 МБ вільного місця;

– звукова карта: Сумісна з DirectX 11.

Рекомендовані вимоги:

– операційна система: Windows 10 (64-біт);

– процесор: Intel Core i5-3470 або еквівалентний;

– оперативна пам'ять: 8 ГБ;

– відеокарта: NVIDIA GeForce GTX 660 або еквівалентна з підтримкою DirectX 11;

– місце на диску: 1 ГБ вільного місця;

– звукова карта: Сумісна з DirectX 11

Дотримуючись даних вимог до технічних та програмних засобів користувачів зможуть успішно користуватись відеогрою без зайвих проблем.

В випадку, якщо відеогра не запускається, будь-ласка перевірте чи ваш персональний комп’ютер або ноутбук відповідає вищеописаним вимогам.

3.6. Висновки до третього розділу

При розробці даного програмного застосунку детально описано реалізацію відеогри, включаючи її основні модулі та системи. Було розглянуто реалізацію механік гравця, супротивників, розглянуто механіку атаки та принцип її роботи. Проведена робота над анімуванням та підключенням анімацій до цих об'єктів. Останнім пунктом було описано графовий метод генерації рівня, який забезпечує випадкову та динамічну побудову ігрових середовищ. Його принцип роботи, приклад роботи та представлено наочно на рисунках, як це відбувається.

Було обрано ручне тестування для імітації реальної діяльності гравця. Це дозволило ефективніше шукати баги та «шпарини» у грі, оскільки тестер міг безпосередньо взаємодіяти з усіма аспектами гри та відстежувати аномалії у поведінці об'єктів та механік.

Тестування охопило всі ключові механіки та системи гри, включаючи переміщення гравця, атаки, отримання шкоди супротивником, відкидання супротивника, "смерть" супротивника, генерацію рівня та колізії об'єктів. Результати тестування були успішними, всі підмодулі працювали коректно, що підтверджує надійність реалізованих механік.

Було складено детальне керівництво користувача, яке включає інструкції по встановленню гри, запуску та управлінню гравцем. Користувач може легко завантажити, інсталювати та запустити гру, дотримуючись вказаних інструкцій.

Також було надано список необхідного програмного забезпечення. Ці вимоги забезпечують належну продуктивність.

Таким чином, у третьому розділі було комплексно розглянуто всі аспекти розробки, тестування та використання гри. Це забезпечує повне розуміння створення та функціонування гри, а також гарантує її якість та стабільність у роботі під час гри.

ВИСНОВКИ

Розробка відеоігор у жанрі «rogue-lite» вимагає розуміння основ цього жанру, що поєднує елементи «rogue-lite» і сучасних підходів до геймплею. Дослідження характеристик, структури та механік таких ігор допомагає створити продукт, який буде цікавим і затребуваним серед гравців. Використання «Unity» та інших сучасних технологій полегшує процес розробки та забезпечує високу якість гри.

Перший розділ був присвячений аналізу ігор у жанрі «rogue-lite» та «rogue-lite». Вивчення їхніх основних характеристик допомогло визначити, що робить ці ігри захоплюючими. Особливу увагу було приділено «спонтанності та структурованості», що важливо для розуміння ігрових механік.

При розробці програмного застосунку увага буле зосереджена на визначенні гри, як «вільного руху всередині закритої структури», що допомагає краще зрозуміти, як створюються ігрові світи і, як гравці взаємодіють з ними. Було розглянуто, як структурувати гру, щоб зробити її цікавою для гравців, включаючи сценарії, процеси та користувацький досвід.

Ігри «Pixel Dungeon» та «Enter the Gungeon» показують, як можна поєднати стиль і геймплей для створення унікального продукту. Було визначено основні вимоги до майбутньої гри, що дозволяє створити якісну відеогру навіть за обмежених ресурсів і коротких термінів розробки. Цей розділ підкреслює важливість адаптації механік та дизайну для створення захоплюючого досвіду.

Другий розділ описує підготовку до створення гри, включаючи проектування архітектури та структури системи, написання псевдокоду та створення схем взаємодії компонентів. Це допомогло чітко визначити основні елементи системи та їх взаємодію, що важливо для успішної розробки гри.

Створення макетів користувацького інтерфейсу допомогло сформулювати принципи якісного дизайну. Було важливо використовувати знайомі елементи інтерфейсу, уникати перевантаження інформацією і вибирати спокійні

кольорові схеми, щоб забезпечити зручність та привабливість інтерфейсу для користувачів всіх типів.

Аналіз інструментів для розробки привів до вибору «Unity», як основної платформи завдяки її потужним можливостям та підтримці мови програмування C#. Використання інкрементної моделі розробки дозволяє поетапно додавати нову функціональність, забезпечуючи гнучкість та можливість своєчасного виявлення та виправлення помилок.

Третій розділ описує процес реалізації гри, включаючи розробку основних модулів та систем. Реалізація механік гравця, супротивників, атак та анімацій вимагає знань та чіткого планування. Було впроваджено метод генерації рівнів, який забезпечує випадкову та динамічну побудову ігрових середовищ.

Ручне тестування гри дозволило виявляти баги та недоліки, оскільки тестер міг безпосередньо взаємодіяти з усіма аспектами гри та відстежувати аномалії у поведінці об'єктів та механік. Тестування охоплювало всі ключові механіки гри, включаючи переміщення гравця, атаки, отримання шкоди супротивником, генерацію рівнів та колізії об'єктів. Результати тестування показали, що всі підмодулі працюють правильно, підтверджуючи їх надійність.

Було складено детальне керівництво користувача, яке включає інструкції з встановлення, запуску та управління грою, що забезпечує легкість у користуванні та високу задоволеність користувачів.

Аналіз та підготовка до створення гри у жанрі «rogue-lite» дозволили визначити ключові аспекти, які необхідно враховувати для створення якісного продукту. Розуміння специфіки жанру, використання сучасних інструментів розробки та детальне тестування забезпечують високу якість гри та задоволення потреб користувачів. Використання інкрементної моделі розробки та чітко спланованої архітектури системи дозволяє ефективно управляти процесом розробки та досягати поставлених цілей у встановлені терміни.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. A brief history of games : вебсайт. URL: https://www.interaction-design.org/literature/article/a-brief-history-of-games (дата звернення: 01.03.2024).
2. TWO THOUSAND YEARS OF BOARD GAMES: вебсайт. URL: https://www.english-heritage.org.uk/learn/histories/board-games/ (дата звернення: 01.03.2024).
3. The full history of board game: вебсайт. URL: https://medium.com/@peterattia/the-full-history-of-board-games-5e622811ce89 (дата звернення: 01.03.2024).
4. Tabletop Games: Historical Evolution: вебсайт. URL: https://corvusbelli.com/en/press-room/blog/games-evolution (дата звернення: 01.03.2024).
5. Tabletop Games: Historical Evolution: вебсайт. URL: https://corvusbelli.com/en/press-room/blog/games-evolution (дата звернення: 01.03.2024).
6. What is Game Design: Step-by-step guide : вебсайт. URL: https://whimsygames.co/blog/what_is_game_design_step_by_step/ (дата звернення: 02.03.2024).
7. What is Game Design : вебсайт. URL: https://www.igi-global.com/dictionary/unifying-instructional-game-design/11822 (дата звернення: 02.03.2024).
8. What is Game Design? What Does a Game Designer Do? : вебсайт. URL: https://www.recepemreercetin.com/vr-game/what-is-game-design-what-does-a-game-designer-do/ (дата звернення: 02.03.2024).
9. 10 Books for Game Designers : вебсайт. URL: https://www.gamedesigning.org/game-design-books/ (дата звернення: 02.03.2024).
10. Jesse Schell. CRC Press; 1st edition (August 4, 2008) The Art of Game Design: A Book of Lenses, 1-288p.

11. Філософія гри Фрідріха Шиллера : вебсайт. URL: https://periodicals.karazin.ua/thcphs/article/view/2386/2129 (дата звернення: 03.03.2024).

12. The Project Gutenberg eBook, The Life of Reason, by George Santayana: вебсайт. URL: https://www.gutenberg.org/files/15000/15000-h/15000-h.htm (дата звернення: 04.03.2024).

13. Ernest Adams and Joris Dormans. New Riders; 1st edition (5 July 2012) Game Mechanics: Advanced Game Design (Voices That Matter), 1-434p.

14. Defining Game Mechanics: вебсайт. URL: https://www.gamestudies.org/0802/articles/sicart#:~:text=Game%20mechanics%20are%20used%20to,that%20encompasses%20all%20these%20aspects. (дата звернення: 06.03.2024).

15. Game mechanics: what are they and what are the most common: вебсайт. URL: https://www.tokioschool.com/en/news/game-mechanics-what-are/ (дата звернення: 06.03.2024).

16. The ULTIMATE begginer’s guide to game mechanics: вебсайт. URL: https://www.juegostudio.com/blog/the-ultimate-beginners-guide-to-game-mechanics (дата звернення: 07.03.2024).

17. Game Mechanics: вебсайт. URL: https://medium.com/@davengdesign/game-mechanics-3f2b338047aa (дата звернення: 08.03.2024).

18. Video Game Mechanics: A Beginner’s Guide (with Examples): вебсайт. URL: https://gamedesignskills.com/game-design/video-game-mechanics/ (дата звернення: 10.03.2024).

19. Roguelike vs. Roguelite: What's the Difference?: вебсайт. URL: https://screenrant.com/roguelike-roguelite-difference-permadeath-hades-rogue-slay-spire/ (дата звернення: 10.03.2024).

20. What Are Roguelike and Roguelite Video Games?: вебсайт. URL: https://www.makeuseof.com/what-are-roguelike-and-roguelite-video-games/ (дата звернення: 11.03.2024).

21. Roguelike Games: 5 Elements of Roguelike Video Games: вебсайт. URL: https://www.masterclass.com/articles/roguelike-game-guide (дата звернення: 12.03.2024).

22. Міроник М.В., Праворська Н.І. Ігровий застосунок в жанрі «Rogue-lite» з використанням технології Unity. Збірник наукових праць за матеріалами XV Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2023». Хмельницький. 2023. – 197-200 с.

23. Upcoming Roguelikes To Keep Your Eye On In 2024: вебсайт. URL: https://www.thegamer.com/best-upcoming-roguelikes/ (дата звернення: 13.03.2024).

24. [2023 Update]How To Choose The Best Gaming Engine For Your Game?: вебсайт. URL: https://pinglestudio.com/blog/co-development/how-to-choose-the-best-gaming-engine-for-your-game (дата звернення: 13.03.2024).

25. The pros and cons of the Unity game engine: вебсайт. URL: https://ventionteams.com/blog/unity-pros-and-cons (дата звернення: 13.03.2024).

26. C++ or C#: What Are and Their Pros and Cons: вебсайт. URL: https://www.sololearn.com/blog/c-plus-plus-or-c-sharp/ (дата звернення: 15.03.2024).

27. Incremental Development: вебсайт. URL: https://deepsource.com/glossary/incremental-development#:~:text=Incremental%20development%20is%20a%20method,designed%2C%20built%20and%20tested%20independently. (дата звернення: 17.03.2024).

28. Input Action System Unity: вебсайт. URL: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/manual/Actions.html. (дата звернення: 5.04.2024).

29. Animation Controller Unity: вебсайт. URL: https://docs.unity3d.com/Manual/class-AnimatorController.html (дата звернення: 7.04.2024).

30. Level Procedural Generator, Ondrej Nepozitek: вебсайт. URL: https://github.com/OndrejNepozitek/Edgar-Unity (дата звернення: 10.04.2024).

31. What is the game testing process?: вебсайт. URL: https://pinglestudio.com/blog/game-testing/what-is-the-game-testing-process (дата звернення: 15.04.2024).

32. Cyberpunk 2077 Bugs: The Biggest Performance Issues So Far: вебсайт. URL: https://www.denofgeek.com/games/cyberpunk-2077-bugs-performance-issues-ps4-ps5-xbox-series-x-stadia-pc/ (дата звернення: 17.04.2024).

33. Video Game Testing Guide: вебсайт. URL: https://qatestlab.com/assets/Guide-Game.pdf (дата звернення: 20.04.2024).

34. A Complete Guide to Game Testing — Its Types and Processes: вебсайт. URL: https://medium.com/@abhaykhs/a-complete-guide-to-game-testing-its-types-and-processes-fc5e3bc66796 (дата звернення: 20.04.2024).

35. What is Usability Testing?: вебсайт. URL: https://www.leanlabeducation.org/blog/usability-testing-for-edtech-companies?gad_source=1&gclid=Cj0KCQjw0_WyBhDMARIsAL1Vz8swNNlHUiyHRmj3A1LDnRqRLnhYgdKSjT1G3tOYZAhn8Hfu3cTBvnkaAnNuEALw_wcB (дата звернення: 20.04.2024).

36. Manual Testing Tutorial for Beginners: вебсайт. URL: https://www.browserstack.com/guide/manual-testing-tutorial (дата звернення: 20.04.2

ДОДАТОК А

PlayerController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEditorInternal;
using UnityEngine;
using UnityEngine.InputSystem;
public class PlayerController : MonoBehaviour
{
    public bool FacingLeft { get { return facingLeft; } }
    public static PlayerController Instance;
    [SerializeField] private float moveSpeed = 1f;
    [SerializeField] private float dashSpeed = 4f;
    [SerializeField] private TrailRenderer myTrailRenderer;
    private PlayerControls playerControls;
    private Vector2 movement;
    private Rigidbody2D rb;
    private Animator myAnimator;
    private SpriteRenderer mySpriteRender;
    private float startingMoveSpeed;
    private bool facingLeft = false;
    private bool isDashing = false;
    private void Awake()
    {
        Instance = this;
        playerControls = new PlayerControls();
        rb = GetComponent<Rigidbody2D>();
        myAnimator = GetComponent<Animator>();
        mySpriteRender = GetComponent<SpriteRenderer>();
    }
    private void Start()
    {
        playerControls.Combat.Dash.performed += _ => Dash();
        startingMoveSpeed = moveSpeed;
    }
    private void OnEnable()
    {
        playerControls.Enable();
    }
    private void Update()
    {
        PlayerInput();
    }
    private void FixedUpdate()
    {
        AdjustPlayerFacingDirection();

        Move();
    }
    private void PlayerInput()
    {
        movement = playerControls.Movement.Move.ReadValue<Vector2>();
        myAnimator.SetFloat("moveX", movement.x);
```

```
        myAnimator.SetFloat("moveY", movement.y);
    }
    private void Move()
    {
        rb.MovePosition(rb.position + movement * (moveSpeed * Time.fixedDeltaTime));
    }
    private void AdjustPlayerFacingDirection()
    {
        Vector3 mousePos = Input.mousePosition;
        Vector3 PlayerScreenPoint = Camera.main.WorldToScreenPoint(transform.position);
        if (mousePos.x < PlayerScreenPoint.x)
        {
            mySpriteRender.flipX = true;
            facingLeft = true;
        } else
        {
            mySpriteRender.flipX = false;
            facingLeft = false;
        }
    }
    private void Dash()
    {
        if (!isDashing)
        {
            isDashing = true;
            moveSpeed *= dashSpeed;
            myTrailRenderer.emitting = true;
            StartCoroutine(EndDashRoutine());
        }
    }
    private IEnumerator EndDashRoutine()
    {
        float dashTime = .2f;
        float dashCD = .25f;
        yield return new WaitForSeconds(dashTime);
        moveSpeed = startingMoveSpeed;
        myTrailRenderer.emitting = false;
        yield return new WaitForSeconds(dashCD);
        isDashing = false;
    }
}
```

EnemyHealth.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class EnemyHealth : MonoBehaviour
{
    [SerializeField] private int startingHealth = 3;
    [SerializeField] private GameObject deathVFXPrefab;
    [SerializeField] private float knockBackThrust = 15f;
    private int currentHealth;
    private Knockback knockback;
    public Flash flash;
```

```
    private void Awake()
    {
        flash = GetComponent<Flash>();
        knockback = GetComponent<Knockback>();
    }
    private void Start()
    {
        currentHealth = startingHealth;
    }
    public void TakeDamage(int damage)
    {
        currentHealth -= damage;
        knockback.GetKnockedBack(PlayerController.Instance.transform, knockBackThrust);
        StartCoroutine(flash.FlashRoutine());
        StartCoroutine(CheckDetectDeathRoutine());
    }
    private IEnumerator CheckDetectDeathRoutine()
    {
        yield return new WaitForSeconds(flash.GetRestoreMatTime());
        DetectDeath();
    }
    public void DetectDeath()
    {
        if (currentHealth <= 0)
        {
            Instantiate(deathVFXPrefab, transform.position, Quaternion.identity);
            Destroy(gameObject);
        }
    }
}
```

Sword.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Sword : MonoBehaviour
{
    [SerializeField] private GameObject slashAnimPrefab;
    [SerializeField] private Transform slashAnimSpawnPoint;
    [SerializeField] private Transform weaponCollider;
    [SerializeField] private float swordAttackCD = .5f;
    private PlayerControls playerControls;
    private Animator myAnimator;
    private PlayerController playerController;
    private ActiveWeapon activeWeapon;
    private bool attackButtonDown, isAttacking = false;
    private GameObject slashAnim;
    private void Awake()
    {
        playerController = GetComponentInParent<PlayerController>();
        activeWeapon = GetComponentInParent<ActiveWeapon>();
        myAnimator = GetComponent<Animator>();
        playerControls = new PlayerControls();
    }
```

```
private void OnEnable()
{
    playerControls.Enable();
}
private void Start()
{
    playerControls.Combat.Attack.started += _ => StartAttacking();
    playerControls.Combat.Attack.canceled += _ => StopAttacking();
}
private void Update()
{
    MouseFollowWithOffset();
    Attack();
}
private void StartAttacking()
{
    attackButtonDown = true;
}
private void StopAttacking()
{
    attackButtonDown = false;
}
private void Attack()
{
    if (attackButtonDown && !isAttacking)
    {
        isAttacking = true;
        myAnimator.SetTrigger("Attack");
        weaponCollider.gameObject.SetActive(true);
        slashAnim = Instantiate(slashAnimPrefab, slashAnimSpawnPoint.position, Quaternion.identity);
        slashAnim.transform.parent = this.transform.parent;
        StartCoroutine(AttackCDRoutine());
    }
}
private IEnumerator AttackCDRoutine()
{
    yield return new WaitForSeconds(swordAttackCD);
    isAttacking = false;
}

public void DoneAttackingAnimEvent()
{
    weaponCollider.gameObject.SetActive(false);
}
public void SwingUpFlipAnimEvent()
{
    slashAnim.gameObject.transform.rotation = Quaternion.Euler(-180, 0, 0);
    if (playerController.FacingLeft)
    {
        slashAnim.GetComponent<SpriteRenderer>().flipX = true;
    }
}
public void SwingDownFlipAnimEvent()
{
```

```
        slashAnim.gameObject.transform.rotation = Quaternion.Euler(0, 0, 0);
        if (playerController.FacingLeft)
        {
            slashAnim.GetComponent<SpriteRenderer>().flipX = true;
        }
    }
    private void MouseFollowWithOffset()
    {
        Vector3 mousePos = Input.mousePosition;
        Vector3 playerScreenPoint = Camera.main.WorldToScreenPoint(playerController.transform.position);
        float angle = Mathf.Atan2(mousePos.y, mousePos.x) * Mathf.Rad2Deg;
        if (mousePos.x < playerScreenPoint.x)
        {
            activeWeapon.transform.rotation = Quaternion.Euler(0, -180, angle);
            weaponCollider.transform.rotation = Quaternion.Euler(0, -180, 0);
        } else {
            activeWeapon.transform.rotation = Quaternion.Euler(0, 0, angle);
            weaponCollider.transform.rotation = Quaternion.Euler(0, 0, 0);
        }
    }
}
```

ДОДАТОК Б

Рисунок Б.1 – слайд презентації №1

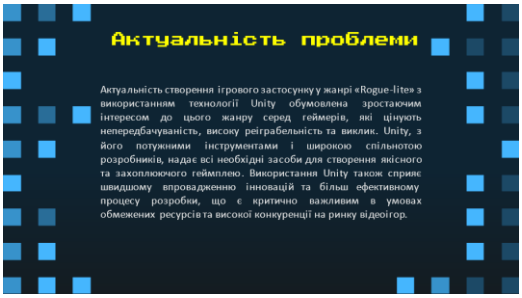


Рисунок Б.2 – слайд презентації №2

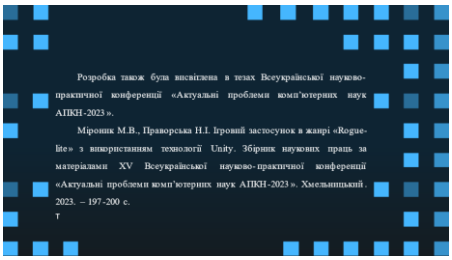


Рисунок Б.3 – слайд презентації №3

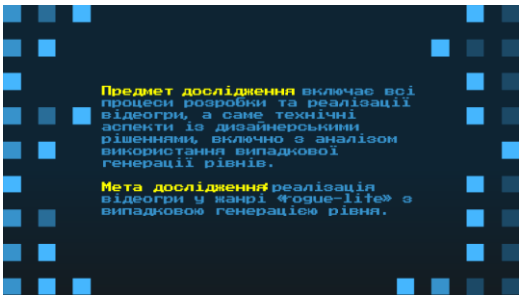


Рисунок Б.4 – слайд презентації №4

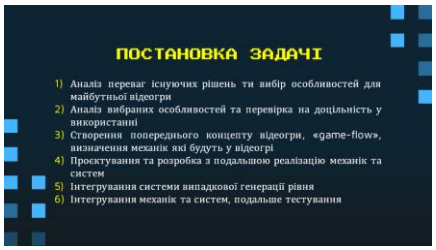


Рисунок Б.5 – слайд презентації №5

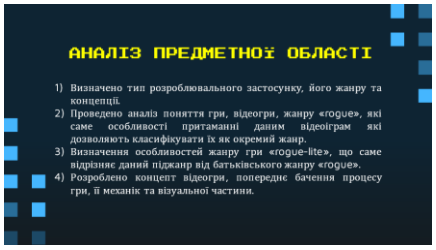


Рисунок Б.6 – слайд презентації №6

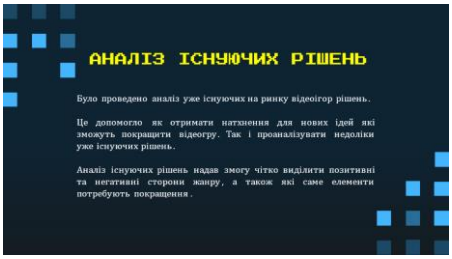


Рисунок Б.7 – слайд презентації №7

Рисунок Б.8 – слайд презентації №8

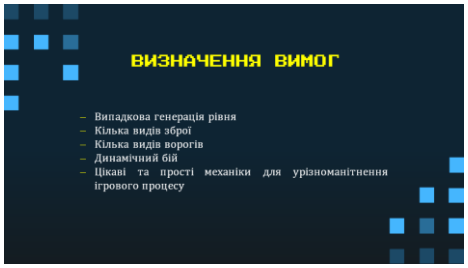


Рисунок Б.9 – слайд презентації №9

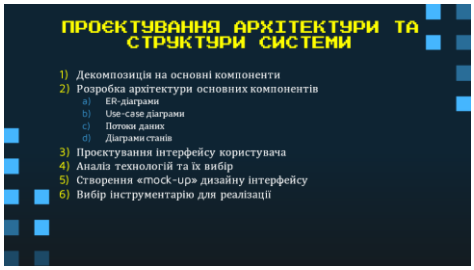


Рисунок Б.10 – слайд презентації №10

Рисунок Б.11 – слайд презентації №11

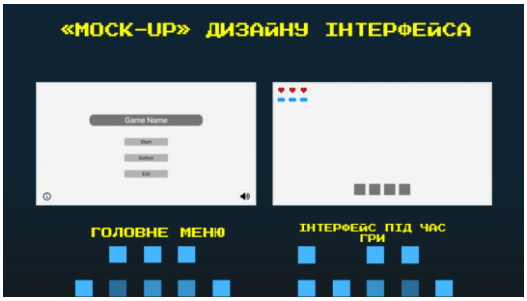


Рисунок Б.12 – слайд презентації №12

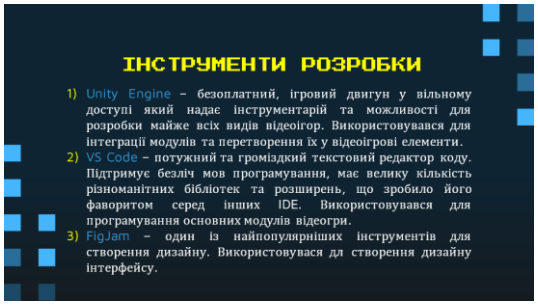


Рисунок Б.13 – слайд презентації №13

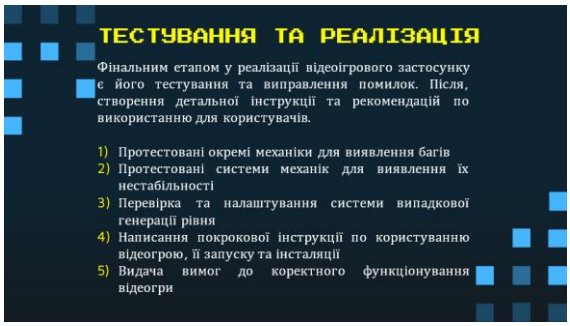


Рисунок Б.14 – слайд презентації №14

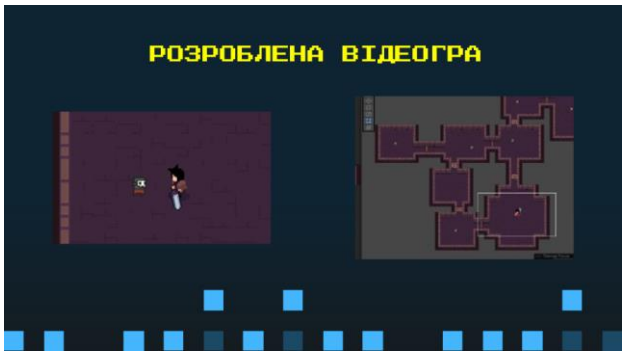


Рисунок Б.15 – слайд презентації №15

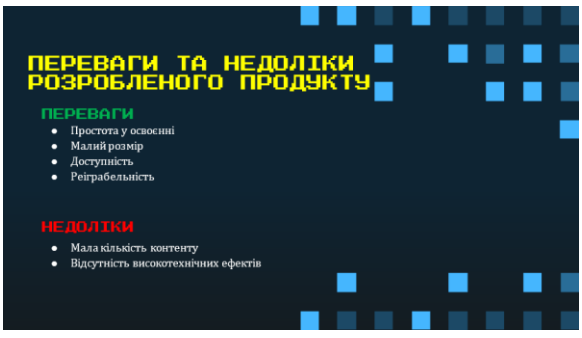


Рисунок Б.16 – слайд презентації №16

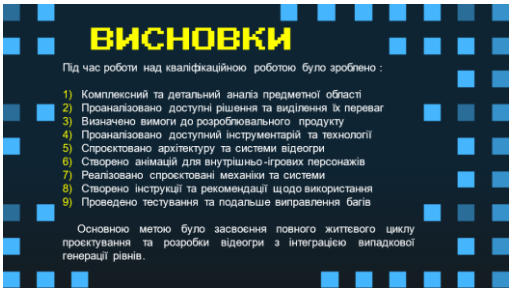


Рисунок Б.17 – слайд презентації №17

Рисунок Б.18 – слайд презентації №18

ДОДАТОК В

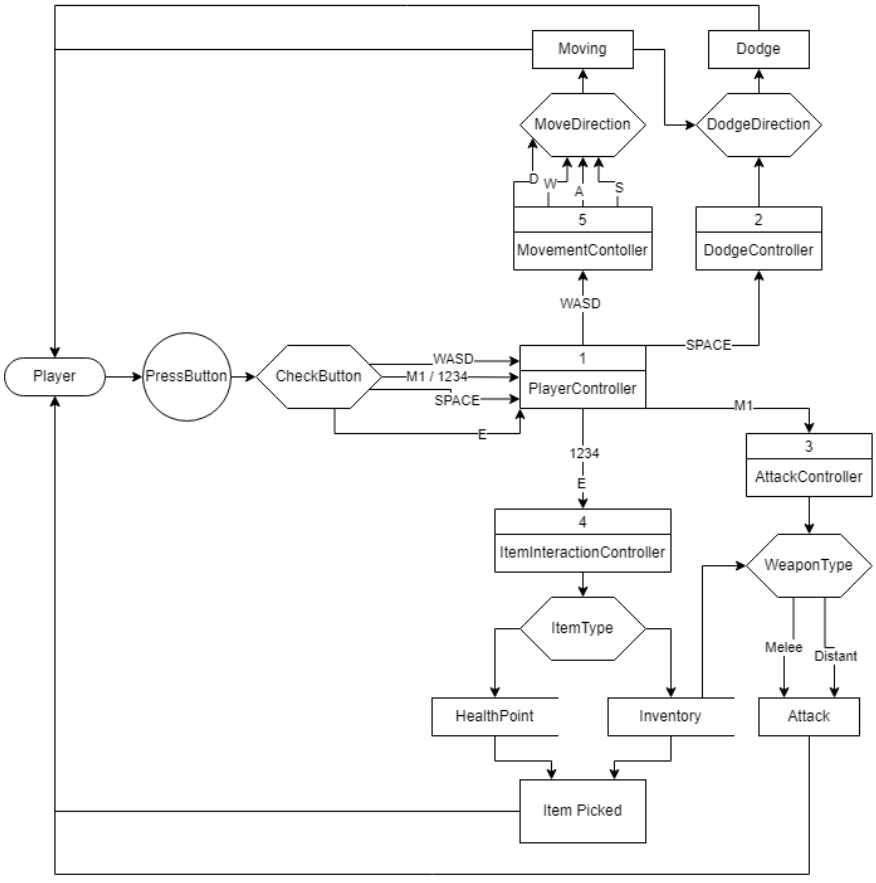
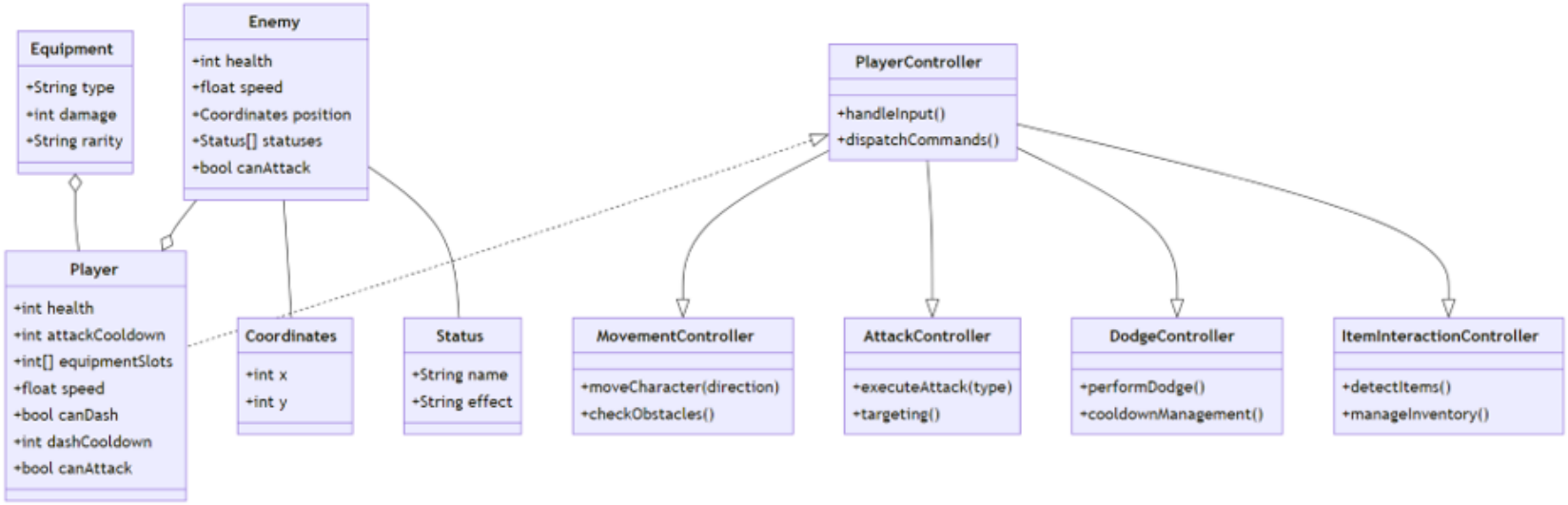
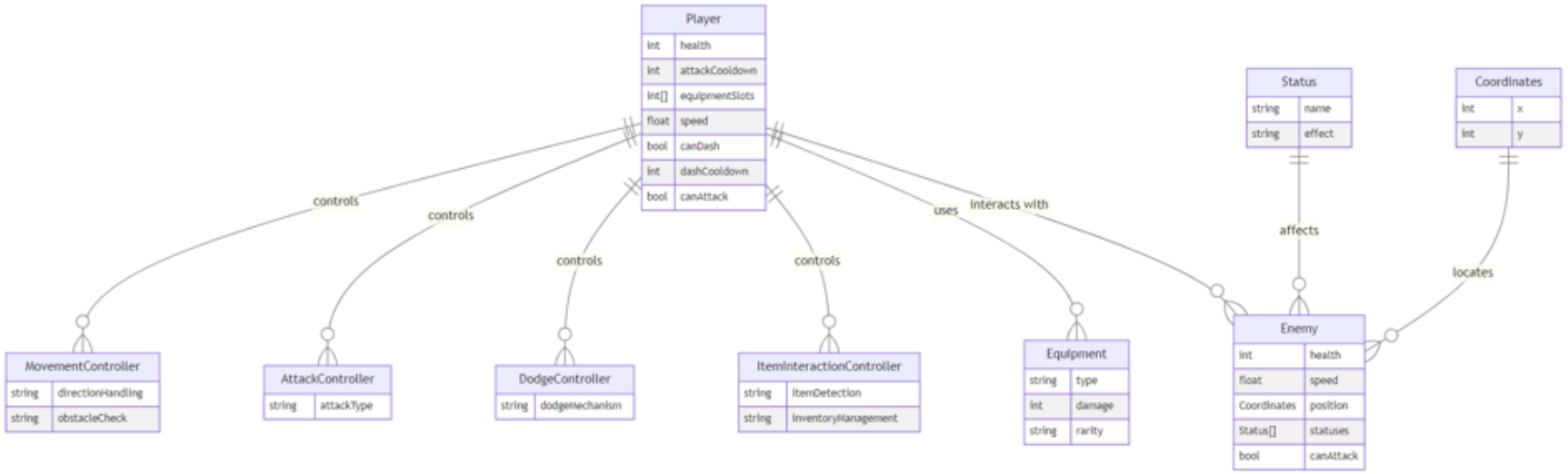


Рисунок В.1 – Діаграма потоків даних

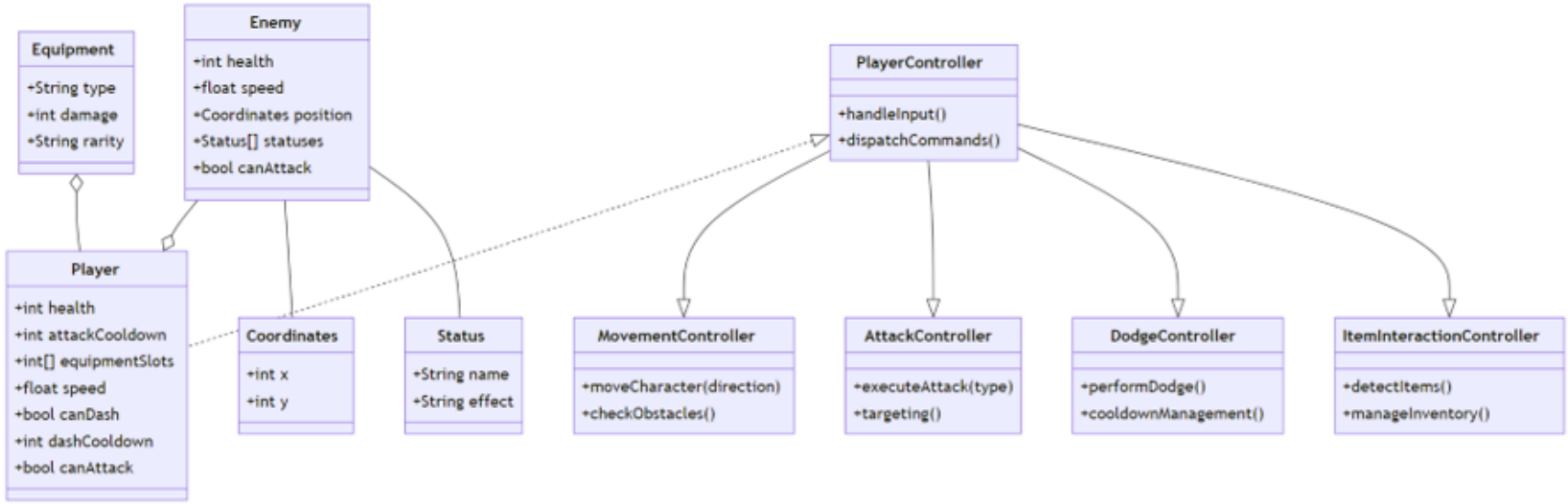
ГРАФІЧНА ЧАСТИНА



					КвРПЗ. 200168.01.16.ПЗ		
					Ігровий застосунок у жанрі «Rogue-lite» з використанням технологій Unity	Літера	Маса	Масштаб
Зм.	Арк.	№ докум.	Підпис	Дата				
Розробив		Мроник М.В.			Відомість документів			
Керівник		Правдорська Н.І.				Аркуш	Аркушів	
Консульт.								
Н. Контр.					UML діаграма прототипів класів			
Зав. каф.		Бебратюк Л.П.						



					КвРПЗ. 200168.01.16.ПЗ	
					Ігровий застосунок у жанрі «Rogue-lite» з використанням технологій Unity	Літера / Маса / Масштаб
Зм.	Арк.	№ докум.	Підпис	Дата	Відомість документів	
Розробив		Мірошник М.В.				Аркуш / Аркушів
Керівник		Правдорська Н.І.				
Консульт.					ER діаграма класів системи гравця	
Н. Контр.						
Зав. каф.		Бебратюк Л.П.				



					КвРПЗ. 200168.01.16.ПЗ		
					Ігровий застосунок у жанрі «Rogue-lite» з використанням технологій Unity	Літера	Маса	Масштаб
Зм.	Арк.	№ докум.	Підпис	Дата				
Розробив		Мірошик М.В.			Відомість документів			
Керівник		Правдорська Н.І.				Аркуш	Аркушів	
Консульт.								
Н. Контр.					UML діаграма прототипів класів			
Зав. каф.		Бебратюк Л.Т.						

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти

Міроник М. В.

Прізвище, ініціали

факультет ІТ, 4 курс, група ІПЗ-20-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності в Хмельницькому національному університеті», згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та/або Anti-Plagiarism) і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

06.06.2024

дата

підпис

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. Поменшли в документах: 15%

ID: 128472 Назва: БКР Ігровий застосунок у жанрі «rogue-lite» з використанням технології Unity Додано в БД: 2024-06-05 Автора: МІРОНЮК Микола Керівники: ПРАВОРСЬКА Н.І., канд. пед. наук, доцент Консультанти: Опоненти:	Документ		Сумарний for no Text Данних	
	Символи	Лексеми	Символи	Лексеми
	99863	950	2716 (3%)	36 (4%)

Джерело плагіату

ID	Опис	Найбільше плагіату в документі	
		Символи	Лексеми

UNICHECK
by Turnitin

Ім'я користувача:
ІПЗ

ID перевірки:
1016316944

Дата перевірки:
04.06.2024 03:03:54 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
04.06.2024 09:00:52 EEST

ID користувача:
100012953

Назва документа: БКР_Ігровий_застосунок_у_жанрі_rogue-lite_з_використанням_технології_Unity_Міроник_М_В_П..

Кількість сторінок: 81 **Кількість слів:** 15424 **Кількість символів:** 119589 **Розмір файлу:** 4.79 MB **ID файлу:** 1016114512

6.03%
Схожість

Найбільша схожість: 2.52% з джерелом з Бібліотеки (ID файлу: 1016111291)

3.97% Джерела з Інтернету	765	Сторінка 83
4.26% Джерела з Бібліотеки	235	Сторінка 88

0.64% Цитат

Цитати	11	Сторінка 89

Не знайдено жодних посилань

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	1

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Міроник Микола Вікторович

Тема Ігровий застосунок у жанрі «Rogue-lite» з використанням технології Unity

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3 ; кількість сторінок записки 75

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі було проведено глибоке дослідження предметної області та аналіз існуючих рішень на ринку. На основі виявлених переваг і недоліків було визначено вимоги до програмного забезпечення. Розглянуто різні технології, розроблено структуру програмного продукту та створено прототип інтерфейсу користувача, які були використані під час розробки програми. Після завершення розробки було проведено низку тестувань для перевірки відповідності вимогам і коректності роботи. Результати тестувань підтвердили готовність програмного забезпечення до експлуатації.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до впроваджених вимог та поставлених завдань.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі було обгрунтовано актуальність теми роботи, визначено мету та основні завдання. У першому розділі детально проаналізовано предметну область, розглянуто всі існуючі рішення з визначенням їхніх переваг і недоліків, на основі чого було сформульовано завдання та вимоги. Також було проведено дослідження відеоігор та геймдизайну жанру «rogue-lite». У другому розділі проведено аналіз і порівняння кількох інструментаріїв, на підставі чого обрано найкращий варіант для цієї роботи — Unity C# з використанням компонентної архітектури та додаткових технологій двигуна. Також розроблено прототип інтерфейсу користувача та структуру проекту, доповнену відповідними діаграмами. У третьому розділі виконано програмну реалізацію з наведенням коду основних модулів, що призвело до створення програмного продукту. Крім того, проведено низку тестувань для перевірки коректності роботи та відповідності вимогам.

4. Позитивні сторони роботи Актуальність теми цієї кваліфікаційної роботи була детально обгрунтована. При аналізі існуючих рішень на ринку програмного забезпечення наведено статистичні дані. Для вибору засобів розробки виконано детальний опис технологій з їхніми перевагами, недоліками та принципами роботи.

5. Негативні сторони роботи Реалізований відеоігровий застосунок може бути покращений за рахунок додання нових механік та систем. Також є актуальним покращити візуальне оформлення застосунку для кращого візуального сприйняття.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано у вигляді різноманітних рисунків та діаграм та відповідають заявленій темі кваліфікаційної роботи. Пояснювальна записка виконана відповідно до вимог та чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує на позитивну оцінку. Матеріал пояснювальної записки структурований, послідовний та детально викладений, що дозволяє легко зрозуміти всю представлену інформацію. Графічні матеріали наочно демонструють всі аспекти виконаної роботи.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана в повному обсязі відповідно до всіх поставлених вимог та завдань і заслуговує на оцінку зарховано

РЕЦЕНЗЕНТ Мартинюк Валерій Володимирович, доктор технічних наук, зав. кафедри автоматизації, комп'ютерно-інтегрованих технологій та робототехніки ХНУ.

"6" 06 2024 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: «Ігровий застосунок у жанрі «Rogue-lite» з використанням технології Unity»

Автор: Міроник Микола Вікторович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Праворська Наталія Іванівна, кандидат технічних наук, доцент
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої й електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системами перевірки на плагіат Unicheck виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, відомість документів), у структурі змісту, назвах розділів/підрозділів тощо, у назвах публікацій у переліку джерел посилання;

2) в якості запозичень системою Unichek було зафіксовано деякі послідовності вихідного коду і посилання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/ схожості, складає 6,03% і адресується до 675 джерел, що, з урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 5.06.2024

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Наталія ПРАВОРСЬКА