МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма: «Розробка програмного забезпечення»

Група: 4РП-07

Дипломний проект

здобувача освіти денної форми навчання
РП.07.15.000.ДП

НЕЙКО
ІЛЛЯ ІГОРОВИЧА

м. Одеса
2024 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **121 «Інженерія програмного забезпечення»**

Освітньо-професійна програма: «**Розробка програмного забезпечення»**

Група: **4РП-07**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту на тему:

Розробка 2D-гри у жанрі роуглайк на програмному рушії Unity

Проектний матеріал складається з пояснювальної записки на 87 сторінках та графічного (презентаційного) матеріалу на 10 аркушах (слайдах)

Дипломник _____ (Нейко І.І.)

Керівник _____ (Джабраілов Д.В.)

Консультанти:

з економічного розділу _____ (Іванченков В.С.)

з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Голова циклової комісії _____ (Кривченко Ю.В.)

Завідувач відділення _____ (Скорнякова О.В.)

Захист «27» 06 2024 р. Протокол ЕК № 3

Оцінка ЕК 4 (добре) / 78 б.

Секретар ЕК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *комп'ютерних систем* Комісія *КТ та ПІ*

Спеціальність *121 «Інженерія програмного забезпечення»*

Освітньо-професійна програма *«Розробка програмного забезпечення»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"15" 01 2024 р.

ЗАВДАННЯ

на дипломний проект

Здобувачеві освіти *Нейко Іллі Ігоровичу*

(прізвище, ім'я, по батькові)

1. Тема проекту *Розробка 2D-гри у жанрі роуглайк на програмному рушії Unity*

затверджена наказом по коледжу від "02" 11 2023 р. № 244-А2-09

2. Термін здачі закінченого проекту 10.06.2024

3. Вихідні данні до проекту *Використання програмного рушія Unity; Використання мови програмування C# та бібліотек Unity для розробки гри; Використання принципів ООП у проектуванні та розробці ігрових проектів; Мають бути реалізовані принципи процедурної генерації ігрового контенту; Реалізація основних ігрових механік комп'ютерної 2D-гри в жанрі роуглайк; Гра повинна мати основні признаки жанру 2D роуглайк; Гра повинна мати інтерфейс.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)

Аналіз ігрового жанру 2D роуглайк; Аналіз принципів процедурної генерації в ігрових проектах; Аналітичний огляд програмних рушіїв з умовно безкоштовним доступом; Формування концепту ігрового процесу 2D-гри в жанрі роуглайк; Проектування основних елементів ігрового процесу та принципи їх роботи; Реалізація основних елементів ігрового процесу; Тестування працездатності ігрових елементів.

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

Особливості ігрового жанру 2D роуглайк; Принципи процедурної генерації в ігрових проектах; Особливості програмного рушія Unity та причини його вибору для розробки проекту; Особливості ігрових механік розроблюємого проекту; Огляд основних елементів ігрового процесу; Принцип роботи основних елементів ігрового процесу; Етапи реалізації основних елементів ігрового процесу; Хід тестування гри.

6. Консультанти по проекту, із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний розділ	*Джабраілов Д.В.*		
Економічний розділ	*Іванченко В.С.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання _____

Керівник *Джабраілов Д.В.* _____
(підпис)

Завдання прийняв до виконання *Нейко І.І.* _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1	*Вступ. Постановка мети та задач проектування*	*20.05.2024*	виконав
2	*Аналіз ігрового жанру 2D роуглайк*	*23.05.2024*	виконав
3	*Аналітичний огляд та вибір програмного рушія*	*25.05.2024*	виконав
4	*Формування концепту ігрового процесу гри*	*28.05.2024*	виконав
5	*Проектування основних елементів ігрового процесу*	*30.05.2024*	виконав
6	*Проектування графічної частини гри*	*01.06.2024*	виконав
7	*Реалізація графічної частини гри*	*03.06.2024*	виконав
8	*Реалізація основних елементів ігрового процесу*	*05.06.2024*	виконав
9	*Імплементація та відлагодження ігрових елементів*	*07.06.2024*	виконав
10	*Тестування працездатності елементів гри*	*09.06.2024*	виконав
11	*Виправлення виявлених помилок*	*10.06.2024*	виконав
12	*Аналіз результатів, підготовка слайдів презентації*	*11.06.2024*	виконав
13	*Економічні розрахунки та питання з охорони праці*	*12.06.2024*	виконав
14	*Підготовка графічної частини проекту*	*13.06.2024*	виконав
15	*Підготовка проекту до захисту та тестування ПП*	*14.06.2024*	виконав

Дипломник _____
(підпис)

Керівник _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			РП 07. 15 000. 00 ДП	Дипломний проект		
А4			РП 07. 15 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	РП 07. 15 000. 00 ДП		
Розробив		Нейко І.І.		10.06	Розробка 2D-гри у жанрі роуглайк на програмному рушії Unity	Літ.	Аркуш 4 / Аркушів 87
Перевірив		Джабраілов Д.В.		10.06		Н Д П	
Н. контр.		Петрашова В.І.		10.06	ВСП "ОТФК ОНТУ" гр.4РП-07		
Затверд.		Кривченко Ю.В.		10.06			

ЗМІСТ

ВСТУП

Індустрія відеоігор є однією з найдинамічніших і найприбутковіших галузей розваг у сучасному світі. За останні десятиліття ігри пройшли шлях від простих аркадних забав до складних інтерактивних творів мистецтва, що поєднують у собі графіку, звук та сюжет. Відеоігри стали важливою частиною життя людей у всьому світі, впливаючи на різні аспекти суспільства від дозвілля до освіти.

Тема дипломної роботи "Розробка 2D-гри у жанрі роуглайк на програмному рушії Unity" відображає сучасні тенденції у світі ігрової індустрії, де ігри роуглайк набувають слави серед геймерів. Ідея створення гри завжди була мені цікава завдяки її унікальним можливостям. Ігри роуглайк пропонують швидкий реіграбельний процес, що забезпечує унікальний досвід кожного разу, коли вони починають нову гру. Цей підхід дозволяє експериментувати з механіками гри, створювати інноваційні системи прогресу та складні випробування для гравців. Зараз роуглайк ігри переживають нову хвилю популярності завдяки інді-розробникам, які зуміли вдихнути нове життя в цей жанр. Вивчення процесу розробки дає зрозуміння основних принципів та прийомів, що можуть бути застосовані в подальших проектах, що робить цю тему дуже актуальною.

Після вивчення всіх можливих варіантів рушії, саме Unity виділявся. Те, що мене зачепило у ньому було широкий набір інструментів та ресурсів, які дозволяють створювати високоякісні ігри. Крім цього, Unity має потужний редактор для створення анімацій, зручну систему управління активами та підтримку сценаріїв на мові C#, що робить його ідеальним вибором для розробки роуглайк гри.

У створеній грі гравці опиниться в світі, де кожен рівень є випадковим. Основними елементами ігрового процесу будуть дослідження підземель, збирання ресурсів та боротьба з ворогами. Також дизайн гри є цікавим та сподобається кожному. У нашому роуглайку смерть персонажа буде остаточною, але кожен новий початок принесе нові можливості та випробування.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Аналіз ігрового жанру 2D роуглайк

Ігровий жанр роуглайк (roguelike) відомий своїми унікальними механіками, які забезпечують високу реіграбельність і непередбачуваність кожної повторної гри. Аналізуючи жанр і прийшовши до висновку, що основні характеристики жанру включають випадково згенеровані рівні, постійні смерті персонажів, покроковий геймилей і складність, яка стимулює гравців до повторних спроб. Також до популяризації цього жанру зробив великий внесок інтерфейс, який був легким та зрозумілим, через що гравці не втрачати цікавість до гри через велику кількість пояснень. Крім того, швидка сутичка з монстрами не обійшла цей жанр стороною, що надавало можливість користувачам не втрачати запал і зразу іти у бій.

1.1.1 Основні характеристики жанру

Однією з ключових рис роуглайк-ігор є випадкове створення рівнів, що робить проходження унікальними та неповторними. Ця механіка досягається за допомогою алгоритмів, які генерують локації, ворогів, скарби та ін. елементи, які заявлені у грі. Ще одна цікава характеристика, яку додали у цього жанрі була постійна смерть (permadeath), бо на відміну від інших жанрів у яких вона була зв’язана з апаратними обмеженнями, або браком пам’яті тут це було зроблено - не випадково. Суть цієї механіки полягає у тому, що відсутня можливість збереження прогресу та якщо персонаж гине, користувач змушений починати гру з самого початку, що додає до ігрового процесу більше емоційної напруги та викликів. У класичних роуглайк-іграх кожна дія персонажа, будь то рух або атака, відповідає відповідному ходу. Це змушує гравців обдумано планувати свої дії, враховуючи поведінку ворогів, наявність скарбів у локації та ін. фактори. Не треба забути, що роуглайк-ігри часто відомі своєю складністю і випадковістю елементами, такими як випадкові зустрічі з ворогами або непередбачувані події, що додають до гри динаміки і в більшості непередбачуваний розвиток подій.

1.1.2 Роуглайк

Перша цифрова гра цього жанру вийшла у 1980 році у вигляді відкритого коду, який був популярний у 1980-х — 1990-х серед студентів. Вона була зроблена для операційної системи Unix і називалась Rogue. До неї виходила схожа гра, під назвою Beneath Apple Manor, але саме Rogue стала першопроходьцем цього жанру та заснувала його. З неї і назва роуглайк (roguelike: схоже на Rogue, або ж така як Rogue). Зазвичай такі ігри того часу були зроблені у сетінгу епічного фентезі, або ж середньовіччя через вплив настільних ігор, в типі Dungeons & Dragons.

Рисунок 1.1. Скріншот гри Rogue

Суть гри була простою треба було дістатися найнижчого рівня гри та дістати таємний амулет Єндора, після чого піднятися на поверхню, але по проходженню гри монстрів у кімнатах ставало все більше та їх складність збільшувалась, а скарбів ставало все менше і менше. Rogue ж в свою чергу показувала нові механіки такі, як: випадкова генерація, постійна смерть, карта з сітки кімнат (утворена випадково). Особливо рандомна генерація внесла основне різноманіття в ігровий процес та зацікавила більшість людей. Саме це зачепило користувача та надало життя жанру і зараз ми можемо багато ігор роуглайк, як і однокористувацьких, так і кооперативних.

1.1.3 Аналіз сучасних 2D роуглайк-ігор

The Binding of Isaac – випущена в 2011 році та розроблена Едмундом МакМіллером, є однією з найвідоміших 2D роуглайк-ігор. Ця гра поєднує у собі елементи роуглайку та шутера з видом зверху, пропонуючи гравцям досліджувати підземелля, які були випадково згенеровані. Кожне проходження цієї гри є унікальне завдяки випадковим рівням, розміщенню ворогів, босів і елементів, саме ці елементи забезпечують високу реіграбельність і змушує гравців пристосовуватись до нових умов кожного разу.

Також генерація цієї гри не є такою простою, бо коридори можуть завести у глухий кут, а інколи можна знайти таємні кімнати. Існують особливі кімнати такі, як: кімнати босів, магазин, бібліотека, кімната жертвоприношення, спавн, ігрова кімната, кімната виклику, янгола, випробування та багато інших. Варіація предметів теж дивує, бо якщо рахувати тільки базову версію гри, то вона налічує близько 700 предметів, які можуть взаємодіяти між собою та впливати на гру. Ці предмети можуть мати різні ефекти, від збільшення здоров’я до зміни способу атаки, що додає глибини та варіації геймплею. Ця гра має сюжет, як основного персонажа, так і для інших (загалом персонажів 7) і для кожного ведеться своя розповідь. Складно не помітити, яку душу та скільки часу вклав автор у кожного з них, бо всі вони відрізняються не тільки історією, а і зовнішнім виглядом. Крім великого різноманіття генерації ця гра може повихвалятись великою кількістю механік. Перетворення, прокляття, сіди. ефекти, політ, теги артефактів та це ще не весь список. Ці механіки створюють цікавий і унікальний ігровий досвід для гравця. Головне знайти тонку границю, де механіки не будуть нагружати гравця і де їх не буде занадто мало.

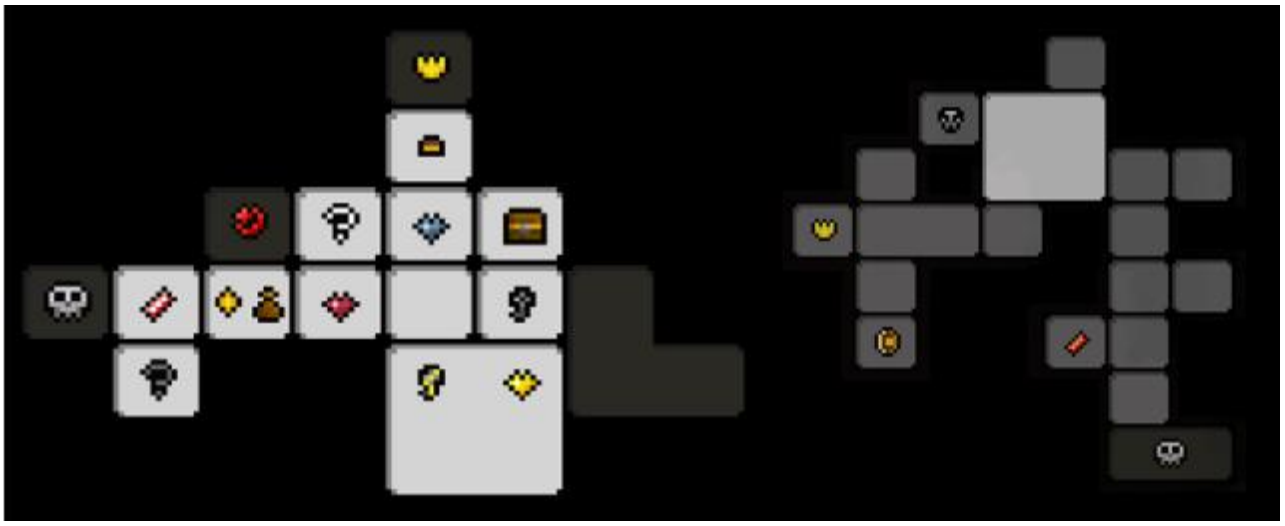


Рисунок 1.2. Скріншот варіації генерації підземель у The Binding of Isaac

Незважаючи на неповторність цієї гри не треба забувати, що вона має і високий рівень складності та можливість розблокування нових предметів та персонажів і закінчень, що стимулює гравців до багаторазового проходження. Гра включає різна режими складності, що дозволяє кожному обрати відповідний рівень виклику.

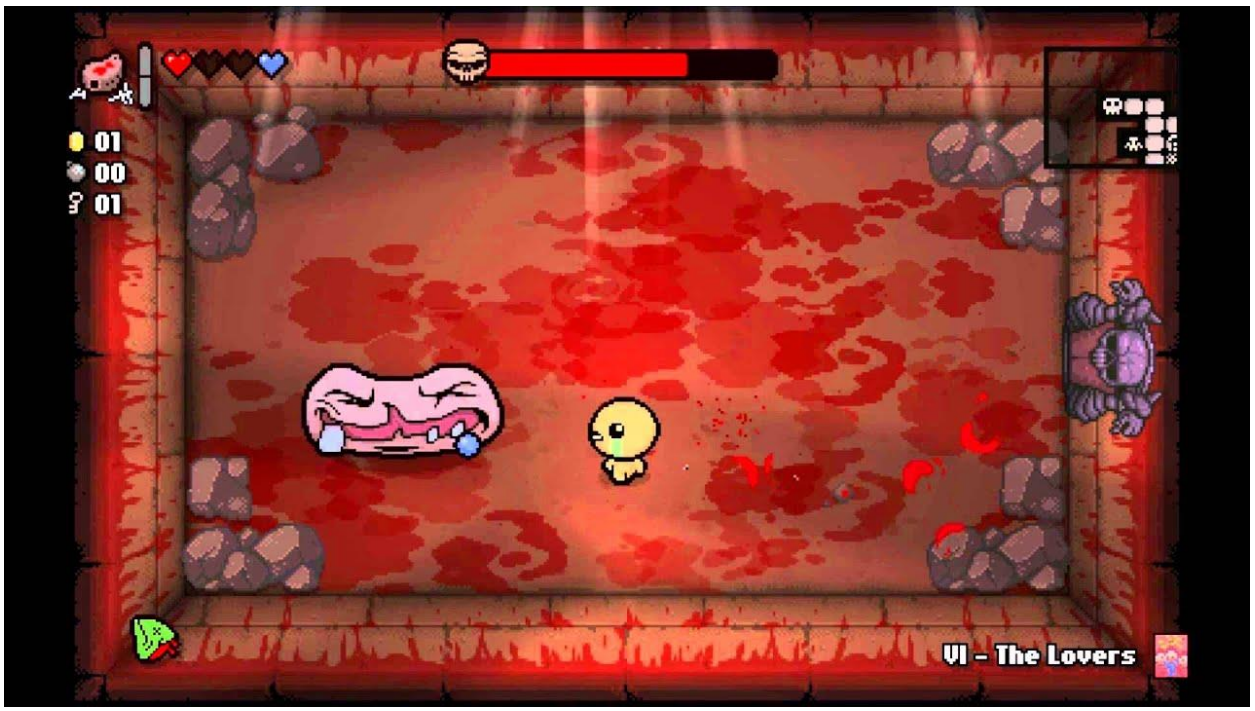


Рисунок 1.3. Скріншот гри The Binding of Isaac

The Binding of Isaac встановила високий стандарт для цього жанру, продемонструвавши, як поєднання простих механік і глибокої варіативності може створити захоплюючий ігровий досвід.

Dead Cells – розроблена Motiom Twin і випущена у 2018 році, є яскравим представником жанру, який поєднує риси роуглайка та метродванії. На відмінно від The Binding Of Isaac ця гра пропонує швидкий та динамічний бойовий процес з різноманітними видами зброї, здібностей. Кожна зброя та навичка має свою унікальну властивість, що дозволяє гравцям створюватирізні комбінації та стратегії ля подолання ворогів. Як і у класичних роуглайках у Dead Cells рівні генеруються випадково, а смерть персонажа змушує гравця починати з початку. Ця механіка додає виклик до гри та стимулює гравців постійно вдосконалювати свої навички. Хоча гра зберігає елементи перманентної смерті, гравці можуть розблоковувати постійні покращення, що додає відчуття прогресу. Це включає нові зброї, навички та інші поліпшення, які залишаються доступними після кожного нового початку гри.

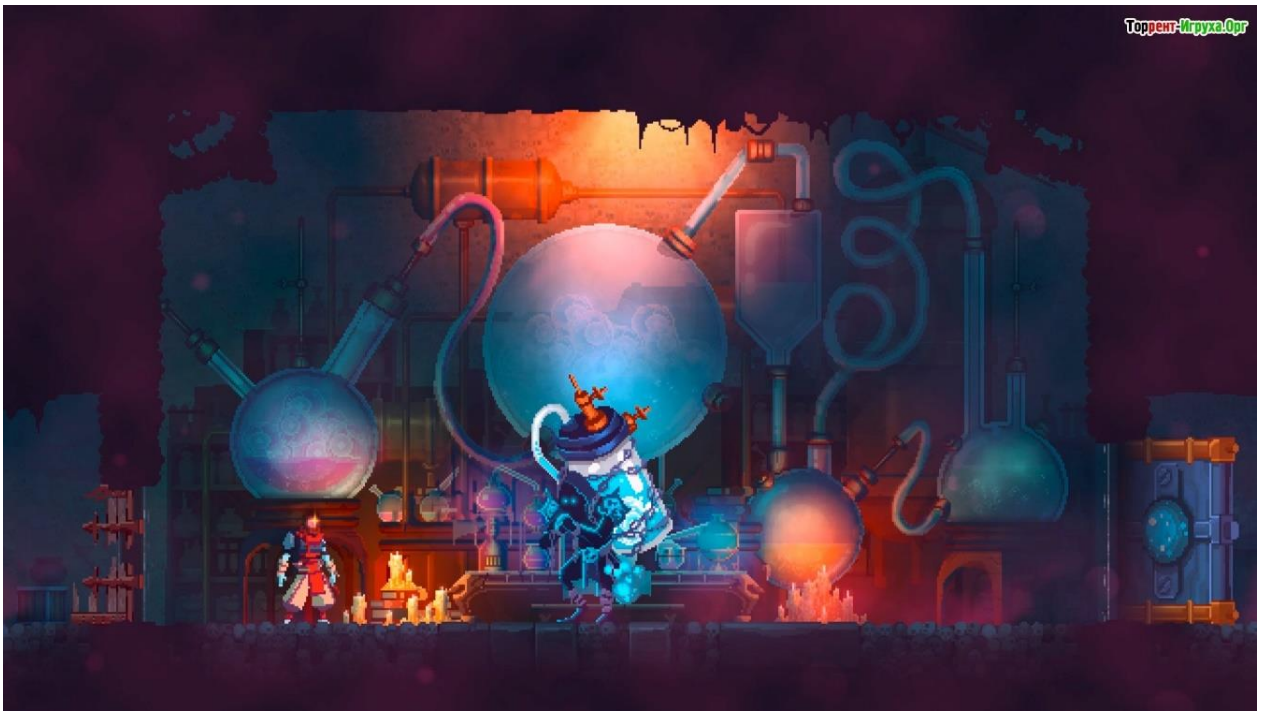


Рисунок 1.4. Скріншот гри Dead Cells

Dead Cells продемонструвала, як можна поєднати швидкий геймплей з глибокими роуглайк-механіками. Гра також вплинула на розвиток жанру, показавши, як можна інтегрувати елементи метродванії в роуглайк-гру для створення багатошарового геймплейного досвіду.

Enter the Gungeon – рохроблено студією Dodge Roll і випущена в 2016 році, стала одним з небагатьох успішних проектів у жанрі 2D роуглайк. Так як і у The Binding Of Isaac гра пропонує досліджувати підземелля наповнені ворогами за зброєю, кожна з яким має унікальні властивості. Графічний стиль гри, натхненний піксельною естетикою ігор того часу, а креативний дизайн ворогів і зброї додають грі особливого шарму. На відмінну від двох інших ігор ця має можливість грати вдвох, що додає грі додатковий шар взаємодії та стратегії. Гравці можуть допомагати один одному у боротьбі з ворогами та босами, що робить гру більш захоплюючою, а головне соціальною. Також механіка яку ми досі не зустрічали – це випадкові події та зустрічі, що роблять кожне проходження непередбачуваним. Поки ми маємо можливість потрапити на такі івенти, як: зустрічі з торговцями, приховані кімнати та ін. сюрпризи, які додають до гри елементи відкриття.

Enter the Gungeon підняла планку для роуглайк-ігор, пропонуючи глибокий і захоплюючий геймплей, який поєднує елементи екшн та стратегічного мислення. Гра також показала, як можна інтегрувати кооперативний режим у роуглайк-гру, зробивши її більш привабливою для широкої аудиторії.

Enter the Gungeon – це must-play для шанувальників роуглайк-ігор та ігор жанру типу bullet hell. Це захоплююча, складна гра, яка пропонує години розваг та реіграбельність, що зацікавлює гравців та вносить різноманіття в ігровий процес.

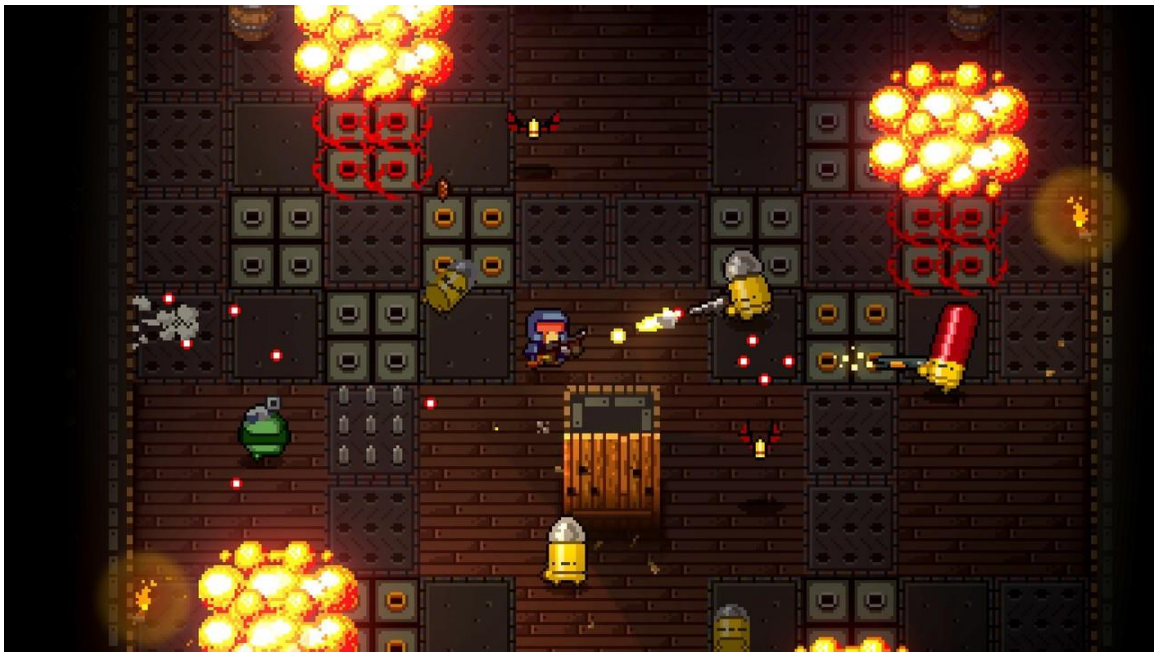


Рисунок 1.5. Скріншот гри Enter the Gungeon

Спільні риси всіх цих ігор це процедурна генерація рівнів, що забезпечує неповторність рівнів, постійна смерть, що не дає можливості продовжити гру після сметрі персонажа, висока складність, яка обумовлена бажанням гравців постіно вдосконалювати свої навички та збирання і використання предметів, які покращують здібності героїв, або надають нові можливості.

Але не можна не помітити відміності цих ігор, які надають родзинку кожній з них. По-перше це різних додатковий жанр, в нашому випадку це – шутер та метродванія. По-друге складно не помітити різні стилістики ігор, наприклад, The Binding Of Isaac має гротескну та темну тематику з глибоким символізмом і розповіддю про внутрішні страхи. Dead cells відзначається стилем метродванії та фокусується на дослідженні нових шляхів, а Enter the Gungeon поєднує кумедний і в якійсь степені абсурдний підхіж до тематики зброї, зі стилізованою піксельною графікою та гумором. Також різною є прогресія і покращення у кожній грі під час проходження. В Dead Cells система включає постійні покращення, які залишаються доступними після початку кожної гри, Enter the Gungeon має ідею розблокування нових видів зброї і предметів під час гри, а в The Binding Of Isaac гравці повинні знаходити нові предмети та розблоковувати інших персонажів по закінчення кожної арки.

Аналіз сучасних 2D роуглайк-ігор показує, що жанр продовжує

розвиватися та залишатися популярним серед гравців. Ігри, такі як The Binding of Isaac, Dead Cells та Enter the Gungeon, демонструють різноманітність підходів до створення роуглайк-геймплею, поєднуючи класичні механіки з інноваційними елементами. Всі вони показують те, що цьому жанру є куди рости і розвиватись

1.2 Аналіз принципів процедурної генерації в ігрових проектах

Процедурна генерація стала невід'ємною частиною новітніх відеоігр, особливо у жанрі роуглайк. Техніка, яка використовується надає змогу створювати різноманітні світи з мінімальними витратами часу, ресурсів на ручну розробку.

Процедурна генерація стала революційною силою в розробці відеоігор, особливо в жанрі роуглайк. Ця потужна техніка дозволяє створювати неймовірно різноманітні світи з мінімальними витратами часу та ресурсів на ручну розробку. Процедурна генерація дозволяє розробникам швидко генерувати цілі світи, рівні, підземелля та інші елементи гри за допомогою алгоритмів і математичних правил.

Це має багато важливих переваг.

1.Нескінченна різноманітність: процедурно згенеровані світи ніколи не повторюються, що робить кожне проходження унікальним. Це робить ігровий процес захоплюючим і свіжим і стимулює повторну гру.

2.Зменшення витрат: Автоматизація процесу побудови світу економить час і ресурси розробників, дозволяючи їм зосередитися на інших аспектах гри, таких як геймплей, історія та дизайн персонажів.

3.Покращена масштабованість: Процедурна генерація ідеальна для створення великих і складних світів, які неможливо створити вручну.

4.Відтворюваність: Гравці можуть проводити години, досліджуючи процедурно згенеровані світи, не повторюючи самих маршрутів чи зіткнень.

1.2.1 Види процедурної генерації

Процедурна генерація часто базується на використанні пседовипадкових чисел для створення різних ігрових елементів. Саме це забезпечує всіма названу унікальність кожного проходження. Один з найпопулярніших алгоритмів, який використовують для створення псевдо-випадкових текстур та ландшавтів є – Шум Перліна. Він дозволяє створювати плавні переходи між різними зонами заради більш реалістичної структури. Все це можливо через велику кількість одиничних векторів через нахил яких вираховується частота шуму в зазначеній точці.

Для генерації гор та печер, зазвичай використовують серединне ділення. Цей метод дозволяє створювати фрактальні патерни з високою деталізацією, а для дерев, рослин та ін. органічних сполук використовують л-системи. Вони базуються на рекурсивних правилах, що дозволяє створювати складні та реалістичні форми.

Використовується метод плиток для створення рівнів. Це працює таких чином, що розміщення передзаданих блоків (плиток) у випадковому порядку дозволяє створювати різноманітні та логічно структуровані рівні. Для більш складних рівнів використовують Вейвлетну матрицю. Вона дозволяє створювати випадкові структури, зберігаючи при цьому закономірності, які потребує.

1.2.2 Переваги процедурної генерації

Рівні, які були процпедурно згенеровані є унікальні та забезпечують неповторне проходження гри, що підвищує інтерес гравців до як умога більшому проходженні. Також вона дозволяє значно скоротити витрати на ручну розробку контенту, а це є дуже важливим для інді-розробників, які тільки починаються свій шлях та мають обмежений бюджет. Процедурна генерація дозволяє створювати динамічні ігрові світи, що реагують на дії гравця і змінюються з часом, а це додає глибини ігровому процесу та підвищує задученість користувачів.

1.2.3 Виклики процедурної генерації

Забезпечення балансу між складністю та ігровою логікою у процедурно згенерованих рівнях є складним, тому випадковість може призвести до створення надто легких, або важких рівнів. Крім цього вона може ускладнювати інеграцію детальних сюжетних елементів, а це актуально для ігор у яких є постійний наратив і глибока історія. Незважаючи на різноманітність, процедурно згенеровані рівні можуть виглядати одноманітно, бо поява цікавого контенту вимагає ретельного налаштування алгоритмів.

1.2.4 Приклади процедурної генерації

У The Binding of Isaac процедурна генерація використовується для ворогів. Предметів та рівнів і забезпечує унікальне проходження. Гра використовує псевдо-випадкові числа для створення різноманітних підземель.

Один з найвідоміших прикладів використання шуму Перлінга є – Minecraft. За допомогою нього створюється світ, який можуть досліджувати і змінувати гравців за бласним бажанням.

А для створення купи планет з різними екосистемами, флорою та фауною використовує процедурну генерацію No Man`s Sky. Він використовує л-системи, що дає неперевершений результат, який приємний оку.

1.3 Аналітичний огляд програмних рушіїв з умовно безкоштовним доступом;

Для виконання дипломного проекту використовувалися ігровий двигун Unity, середовище розробки програмних проектів Visual Studio 2019 та графічний редактор Adobe Photoshop CS6.

Unity є сучасним ігровим двигуном, який постійно оновлюється та розширює свої можливості. На рисунку 1.6 представлена робоча область ігрового двигуна Unity. Цей двигун є одним з найпоширеніших ігрових двигунів, що використовуються для розробки ігор для смартфонів та інших кишенькових пристроїв. Він має зручний та інтуїтивно зрозумілий інтерфейс, який можна налаштувати під свої потреби в залежності від проекту.

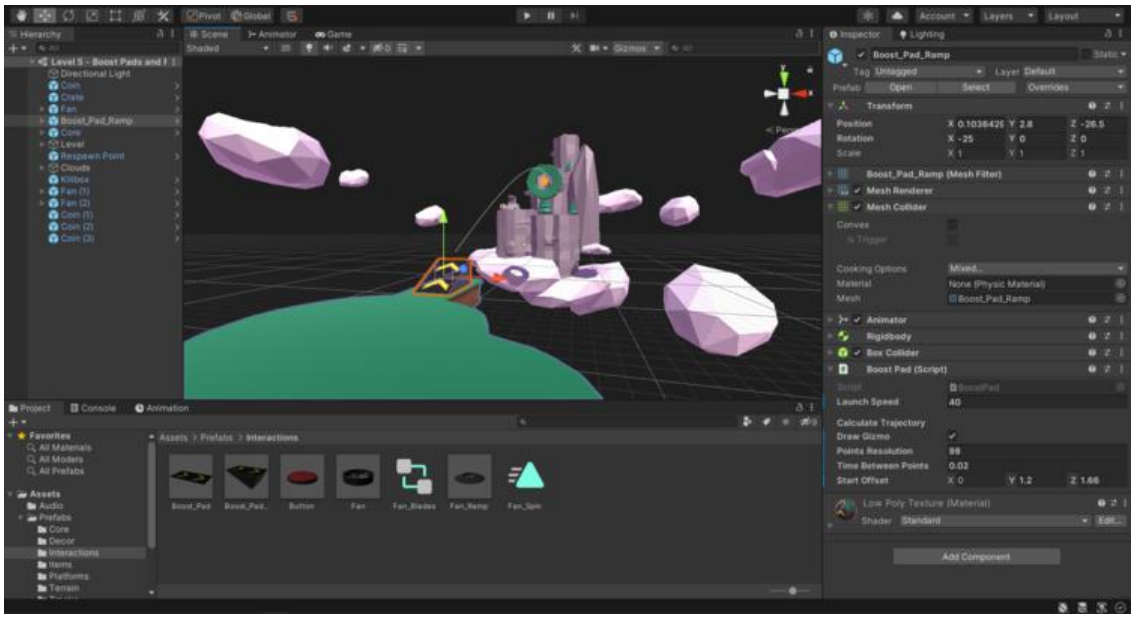


Рисунок 1.6. робоча область ігрового двигуна Unity

Цей ігровий двигун дозволяє працювати у зручному тривимірному просторі редактора сцени. Ви можете розміщувати об'єкти, редагувати їх властивості, компоненти та можливості відповідно до ваших потреб. Крім того, інтерфейс та можливості движка Unity дозволяють зручно працювати над розробкою ігор. Unity має інтерфейс та функціонал, які дозволяють створювати основні типи файлів, необхідні для роботи над проектом, та редагувати їх безпосередньо у редакторі. Крім візуальної складової ігрового проекту, середа розробки також надає можливість редагувати ігровий код за допомогою системи скриптів, які додаються до об'єктів та розширюють їх функціонал. У поточній версії Unity використовується мова програмування C#, що дозволяє використовувати всю потужність цієї мови програмування. Крім стандартних бібліотек C#, Unity має власні бібліотеки, з якими можна працювати у своєму проекті. Доступна та детальна документація на середу розробки допомагає швидко орієнтуватись у можливостях цього ігрового двигуна.

Великим плюсом є умови ліцензування. Unity є безкоштовним для навчання, а також для персональних цілей. Навіть якщо створит игру на їх рушії і розмістити її у магазинах, ліцензію треба буде платити лише в разі річного грошового прибутку більше ніж 100000 доларів. Це дуже зручно для маленьких інді-студій, які лише починають входити у цей ринок. На данний момент, Unity є популярним для створення мобільних ігор, а також деяких

крупних проектів, таких як Holllow Knight(Рисунок 1.7.), Ori and the Blind Forest(Рисунок 1.8.), Subnautica(Рисунок 1.9.), Slime Rancher(Рисунок 1.10.).

Висока популярність Unity також підтверджується його використанням у різних індустріях поза межами ігрового світу, таких як архітектура, візуалізація продуктів, кінематограф та навчальні програми. Це свідчить про його універсальність і адаптивність до різних вимог і сценаріїв використання.

Таким чином, Unity не тільки відповідає сучасним стандартам розробки ігор, але й встановлює нові, завдяки чому продовжує утримувати свою позицію як один з найпопулярніших і найбільш затребуваних рушіїв у світі.

Рисунок 1.7. Скріншот гри Hollow Knight

Рисунок 1.8. Скріншот гри Ori and the Blind Forest

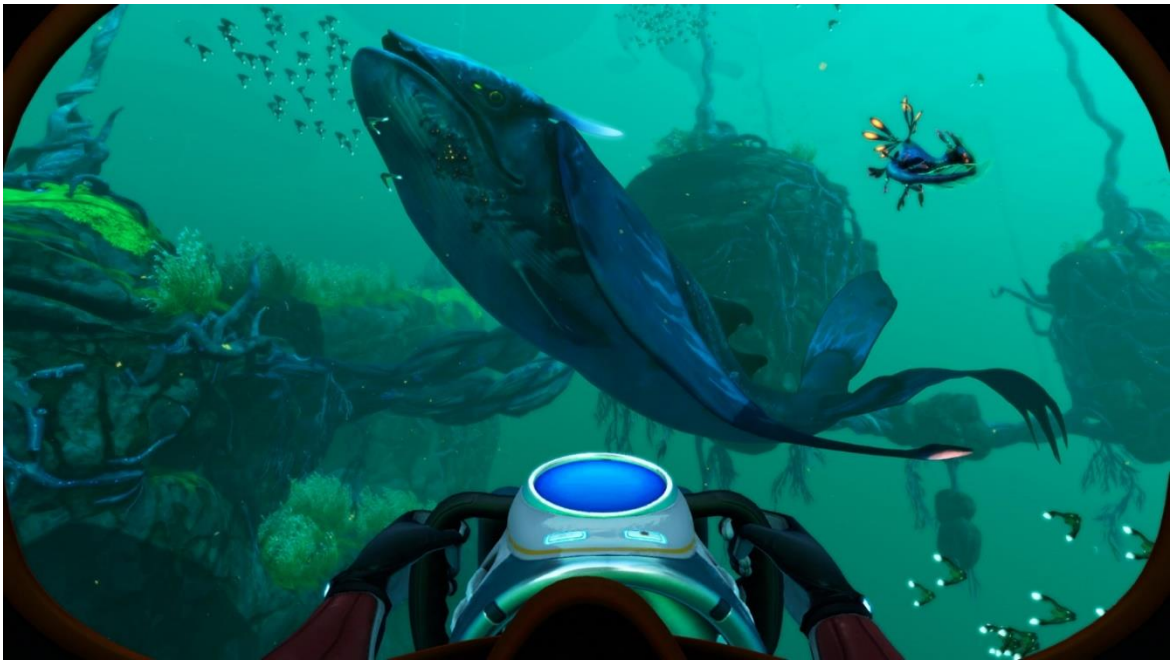


Рисунок 1.9. Скріншот гри Subnautica

Рисунок 1.10. Скріншот гри Slime Rancher

Visual Studio2019 використовується для написання ігрового коду. Visual Studio-це пакет середовища розробки програмного забезпечення від Microsoft. Це середовище розробки дозволяє створювати проекти з використанням різних мов програмування, баз даних, бібліотек та інших програмних рішень. Visual Studio-це зручний редактор коду, що дозволяє переміщатися по проекту, його бібліотекам і редагувати посилання всередині проектів і між ними.

Ліцензія, за якою поширюється Visual Studio, дуже гнучка. Розповсюджується безкоштовно для використання в середовищі розробки в освітніх, наукових, особистих та некомерційних цілях. Крім того, функціональної різниці між безкоштовною та платною версіями немає.

Adobe Photoshop CS6 використовується для створення або редагування растрових зображень. Кількість інструментів для редагування та створення зображень настільки велика, що ви можете виконувати будь-яку роботу, від простих ескізів до створення частин зображення за допомогою нейронних мереж у пізніших версіях програмного забезпечення. Програмне забезпечення розповсюджується лише за ліцензією, але існує безкоштовна пробна версія, яка використовується для створення спрайтів та зображень для ігрових проектів.

1.4 Формування концепту ігрового процесу 2D-гри в жанрі роуглайк

Для виконання поставлених задач дипломного проектування, необхідно виконати повний цикл розробки проекту цифрової гри, який складається з декількох етапів. Це технічний опис гри, де необхідно виконати постановку завдань, а для цього розібрати з яких елементів буде складатись гра.

Оскільки ігровий жанр визначений в темі дипломного проектування, наступним кроком буде виділення загального ігрового концепту. З жанрової точки зору, 2D роуглайк, в першу чергу виражається через особливість проходження рівня – кожен раз генерується новий, неподібний на попередні рівні. Рівень представляє з себе декілька кімнат, які між собою мають двері. Гравець вільно може пересуватись через двері в кімнатах, але лише після того, як вб’є всіх противників в неї.

Окреме питання отримання пошкоджень гравцем. Гравець може отримати пошкодження лише двома типами: в ближньому бої та на дистанції. Перший тип пошкоджень гравець отримує від противників, які б’ють близько та від різних пасток, а другий тип – від противників, які б’ють на дистанції кулями. З іншої сторони, для таких ігор є нагальним питанням розміщення «жетонів» для додаткового життя, тому було реалізовано розміщення на рівні додаткових предметів, які підвищують здоров’я.

Окрім предмета, яке підвищує здоров’я гравця, ще були реалізовані такі предмети: Пояс та Черевик. Перший підвищує швидкість стрільби гравця, а другий – швидкість переміщення.

В грі є 2 типу ворогів: ті, які наносять пошкодження в ближньому бої та ті, які наносять пошкодження на дистанції пулями. Також вороги слідують за гравцем, щоб підвищити складність.

Розглянувши всі ці основні жанрові питання, можна сформувати із них наступні положення, щодо ігрового концепту майбутньої гри:

Гра буде мати реіграбельність. Це означає, що кожна наступна спроба буде новою та відрізнятися;

Гра буде реалізовувати систему предметів, які посилюють статистику гравця;

На рівнях будуть з’являтися об’єкти для підвищення статистики гравця;

На рівнях будуть з’являтись вороги, які будуть переслідувати гравця;

Гравцю необхідно буде протистояти двом типам ворогів.

Розділ гри на рівні та сцену необхідно було реалізовувати виходячи з технічних особливостей ігрового рушія Unity, на якому буде виконуватись робота. Рівень буде реалізовуватись завдяки сцені, на якій будуть генеруватися кімнати, предмети та вороги, знаходитись гравець, його камера.

Система ігрових ресурсів також буде реалізовуватись за допомогою програмних рішень та реалізації об’єкту гравця зі своїми параметрами. Вони будуть пов’язані із статистикою персонажа та зможуть впливати на них.

Відповідати за генерацією об’єктів для зміни показників статистики, ворогів на рівні і сам рівень буде ігровий менеджер, що за допомогою коду буде реалізовувати це. Із загальним концептом ігрового процесу можна ознайомитись на рисунку 1.11.

Червоний куб – це ворог, який буде генеруватися рандомно кожен раз, як буде генеруватися рівень. Синій – це гравець, він завжди буде генеруватися в центрі першої кімнати. Фіолетовий – це предмети, які будуть генеруватися рандомно кожен раз, як буде генеруватися рівень. Це загальний концепт ігрового процесу.

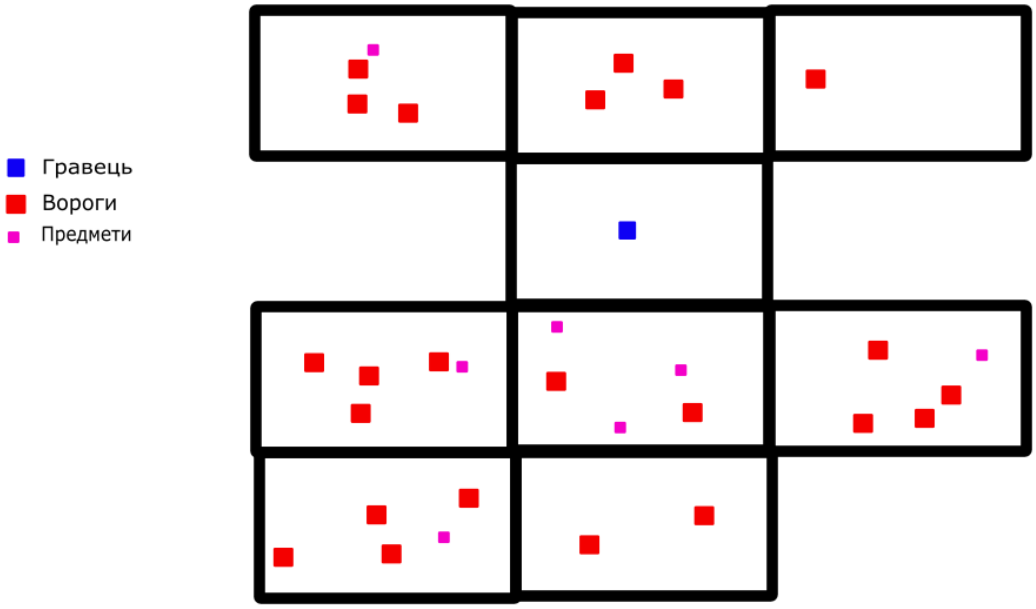


Рисунок 1.11. Загальний концепт ігрового процесу

Окремо треба розглянути питання ігрового інтерфейсу, який буде використовуватися гравцем. Інтерфейс має бути простим та зрозумілим для граця, бо буде відображати декілька показників: Кількість здоров'я у вигляді полоски, кількість здоров'я в цифрах та кількість зібраних предметів. Усі елементи інтерфейсу мають змінюватись в реальному часі, бо залежать від ігрової ситуації, а тому мають бути видними для гравця у любий момент гри. Ці елементи будуть відображатись на краях екрану, які будуть заповнюватись в залежності від стану чи змінювати своє значення. Із загальним концептом ігрового інтерфейсу гри можна ознайомитись на рисунку 1.12

Всі три показника розташовані на канві, що дозволяє відображати все поверх гри, як інтерфейс.

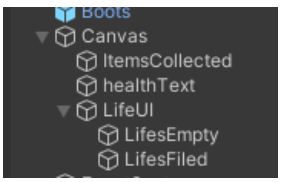


Рисунок 1.12. Структура інтерфейсу в грі

Перше – це кількість зібраних предметів гравцем, друге – це здоров'я в цифрах, а третє – теж здоров'я, але вже у вигляді повзунка, який заповнюється відносно того, скільки здоров'я в гравця.

Рисунок 1.13. Повне та неповне здоров’я гравця

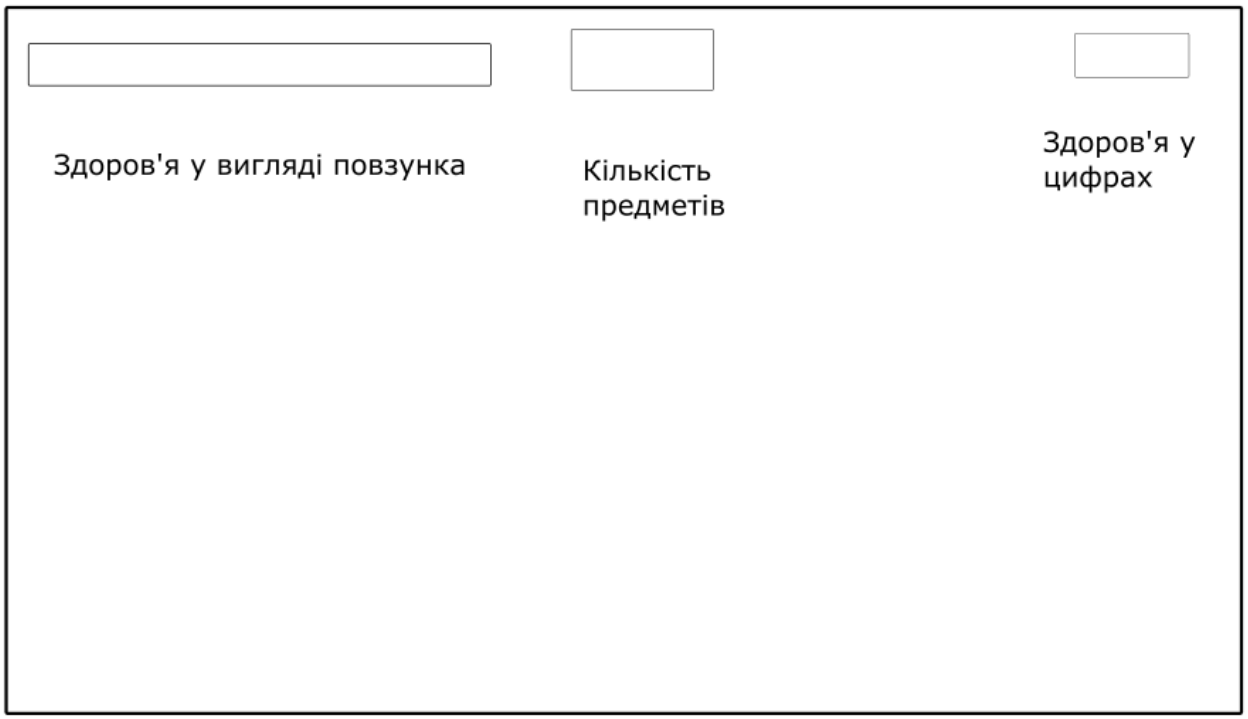


Рисунок 1.14. Загальний концепт ігрового процесу

1.5 Проектування основних елементів ігрового процесу та принципи їх роботи.

Перед початком виконання реалізації проекту, треба виконати проектування основних елементів ігрового процесу та принципи їх дії. Для цього спочатку треба розділити проект на частини. Всього можна виділити наступні основні частини проекту:

Код ворожих об’єктів;

Код гравця;

Код інтерфейсу;

Код ігрових систем.

Кожний із таких змістовних модулів має перелік коду. Схему основних модулів та коду, що належить до нього можна побачити на рисунку 1.15.

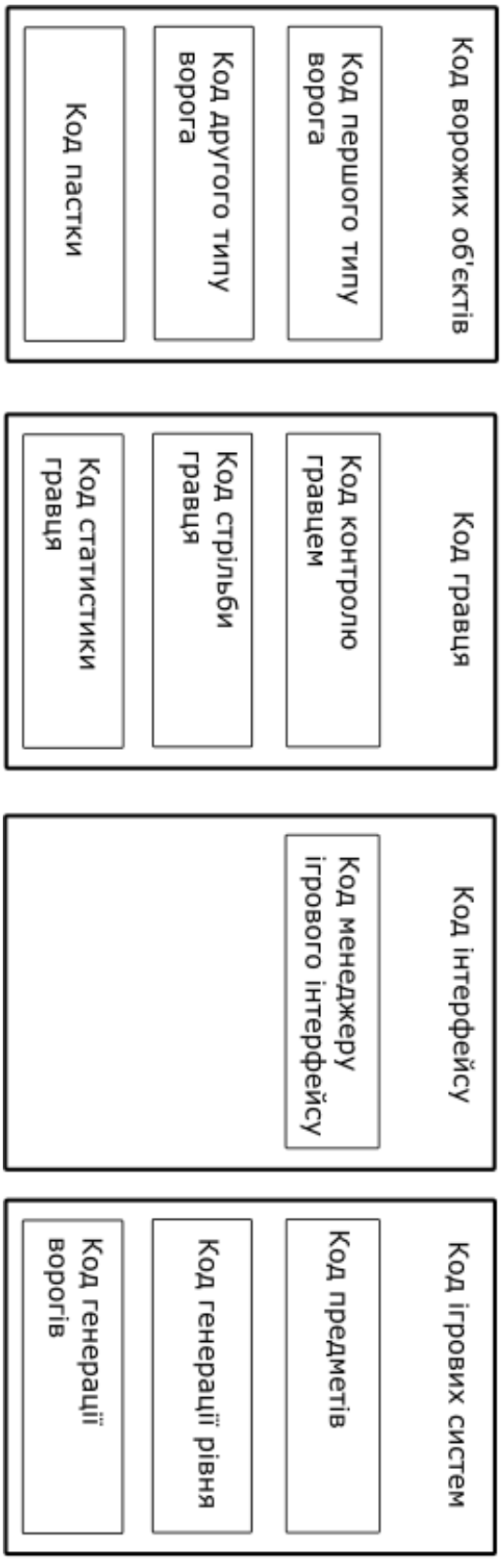


Рисунок 1.15. Схема основних модулів гри та коду, що належить до них

Об’єкт має логічну структуру із коду, який відповідає за його логіку атаки об’єкту, переміщення та поведінки з оточенням.

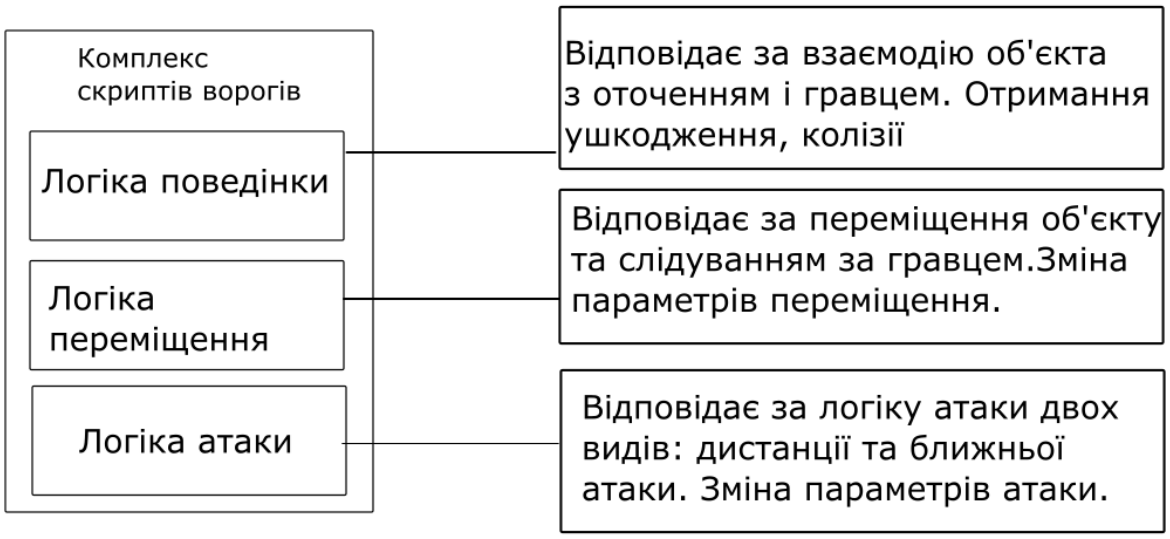


Рисунок 1.16. Схема змісту складових ворожого об’єкту

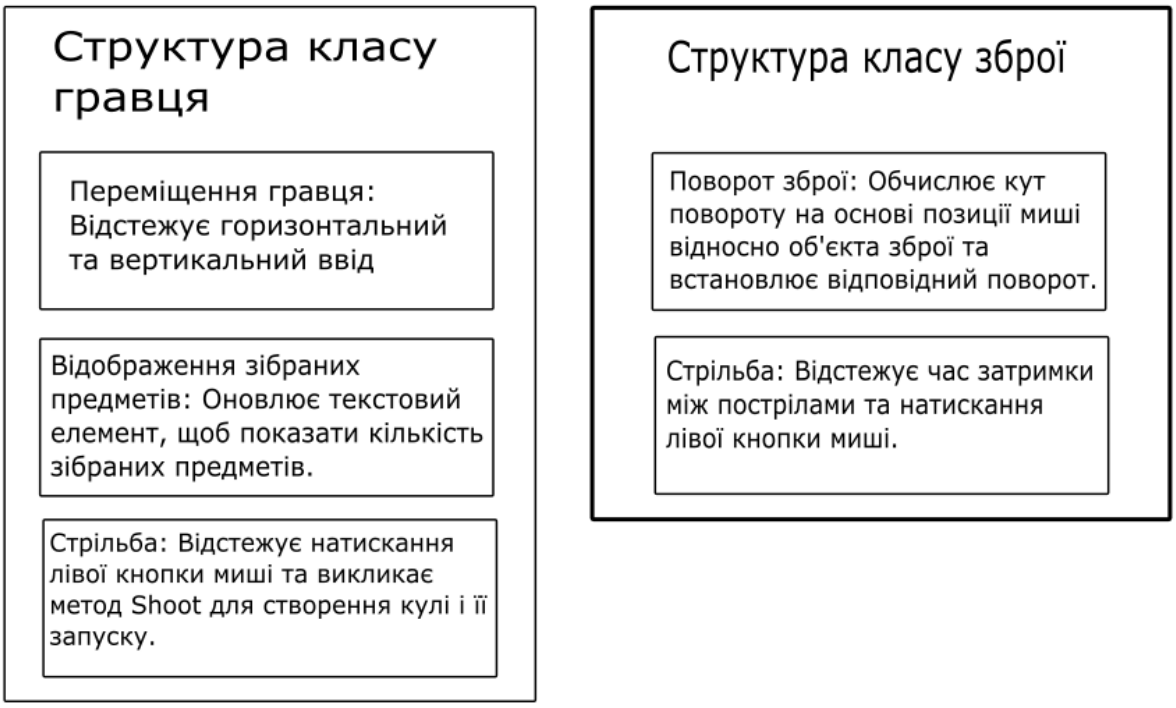


Рисунок 1.17. Схема структури загального класу гравця та зброї

Принцип роботи генерації рівня:

Розглянемо Модуль "Двері" (Door)

Типи дверей: Клас Door визначає типи дверей (ліво, право, верх, низ) за допомогою enum DoorType.

Взаємодія з гравцем: При зіткненні з гравцем (метод OnTriggerEnter2D), гравець переміщується на нову позицію залежно від типу дверей.

Пошук гравця: Під час старту (Start()) скрипт знаходить гравця за тегом "Player".

Розглянемо модуль "Підземелля" (DungeonCrawler, DungeonCrawlerController, DungeonGenerator)

Принцип роботи Crawler

Клас DungeonCrawler описує об'єкт, який може переміщуватися у випадковому напрямку, тип самим створюючи підземелля.

Керування Crawler'ами: DungeonCrawlerController створює та керує кількома DungeonCrawler, відстежуючи їхні відвідані позиції.

Генерація підземелля: Клас DungeonGenerator використовує дані з DungeonGenerationData для генерації підземелля та викликає метод для створення кімнат у відповідних позиціях.

-Модуль "Кімнати" (Room, RoomController)

Принцип роботи:

Кімната: Клас Room описує окрему кімнату підземелля, включаючи її розмір та двері. Він також керує активацією та деактивацією дверей, які не з'єднані з іншими кімнатами.

Керування кімнатами: RoomController відповідає за перевірку наявності кімнат ,завантаження кімнат, збереження їх у черзі, , а також керування переходами між ними. Коли гравець входить у кімнату, цей модуль оновлює стан дверей у кімнаті та ворожих об'єктів.

1.6 Реалізація основних елементів ігрового процесу

Впровадження ключових елементів ігрового процесу є важливим етапом розробки будь-якої гри, це полягає в розробці основних механізмів і систем, які визначають ігровий процес. Цей процес вимагає ретельного планування, проектування та програмування, адже від цього залежить емоційна напруга, динамічність та збалансованість проекту. Саме він визначає, чи буде гра цікавою, захоплюючою та повторюваною.

В основі механізма нашої гри гравець керує персонажем, та його дії безпосередньо залежать від користувача. Тому якісна реалізація ключових елементів геймплея є запорукою успіху проекту. Також важливо зазначити, щоб всі елементи ігрового процесу гармонійно поєднувалися, і не створювали дисбалансу або незрозумілої складності. Саме баланс робить гру веселою та справедливою для всіх гравців. Геймплей роуглайка спрямований на те, щоб гравець був зацікавлений і заінтересований у дослідженні світу та продовженні гри. Важливо вибрати правильний рівень складності, забезпечити різноманітність і створити відчуття прогресу. Через реіграбельність продукту гравці захочуть повертатися до гри знову і знову, для цього важливо дослухатися до ком'юніті, додавати новий контент і створювати умови для підтримки інтересу до проекту.

Реалізація ключових елементів ігрового процесу є складною процедурою, але критичним етапом розробки. Визначивши основні елементи ігрового концепту та їх проектування, є можливість розглянути технічну реалізацію основних елементів ігрового процесу. Це забезпечує створення захоплюючої, якісної та збалансованої гри, яка задовольнить вимоги гравців та забезпечить їм незабутній ігровий досвід.

Для виконання завдання дипломного проектування необхідно реалізувати ряд основних модулів, що були визначені у попередніх розділах. Для цього запускаю рушій та переходжу до вкладки створення проекту. Так як гра двовимірна, то створюємо проект на основі 2D.

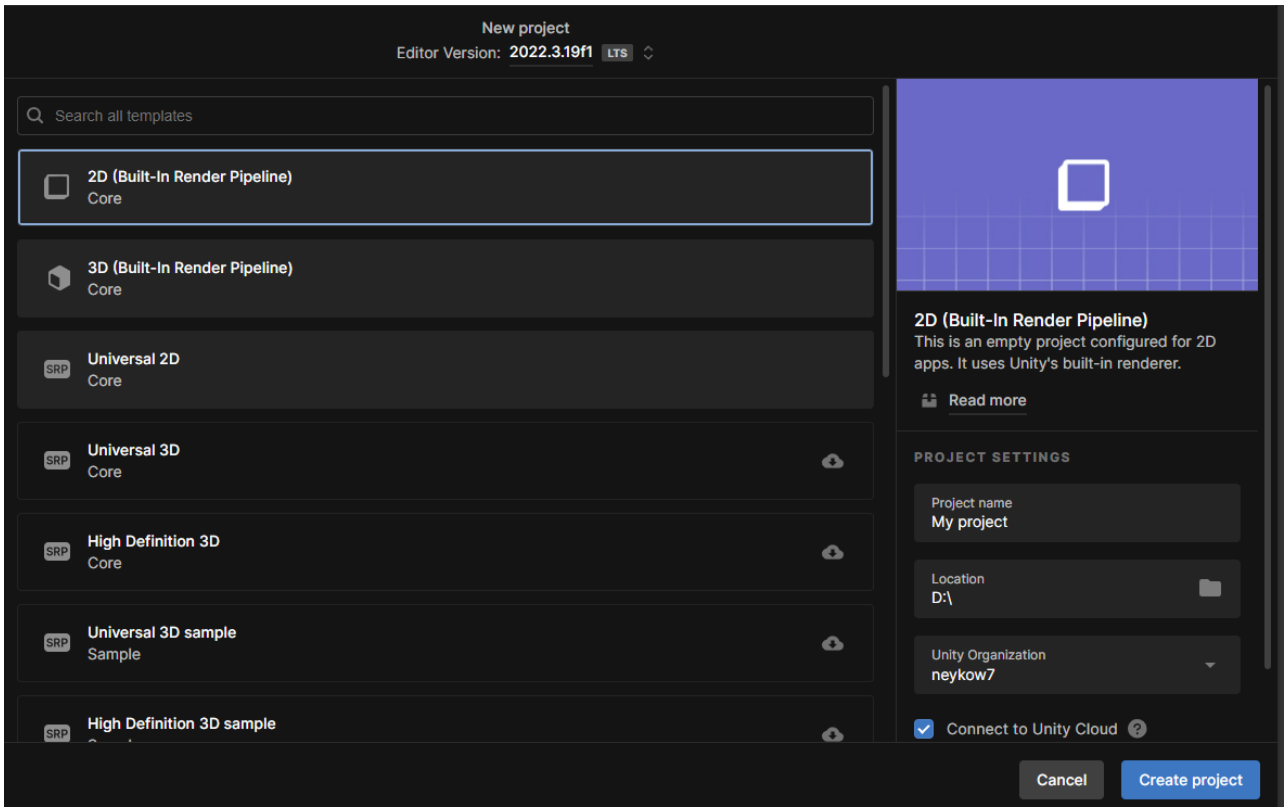


Рисунок 1.18. Створення проекту

Виконавши створення нового проекту, на сцені буде автоматично згенерована пуста сцена проекту, із стандартним набором об’єктів – камери гравця, що можна побачити на рисунку 1.19.

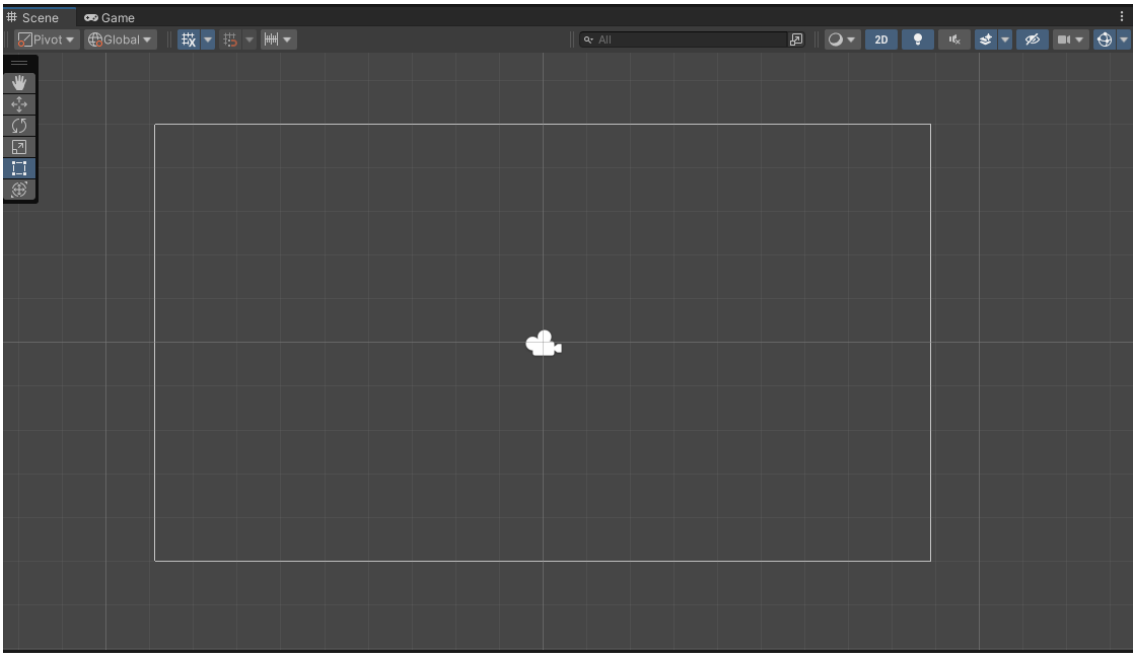


Рисунок 1.19. Стартовий об’єкт в новому проекті Unity

Згідно існуючого концепту інтерфейсу меню, створюємо всі необхідні елементи: показник здоров’я в цифрах.

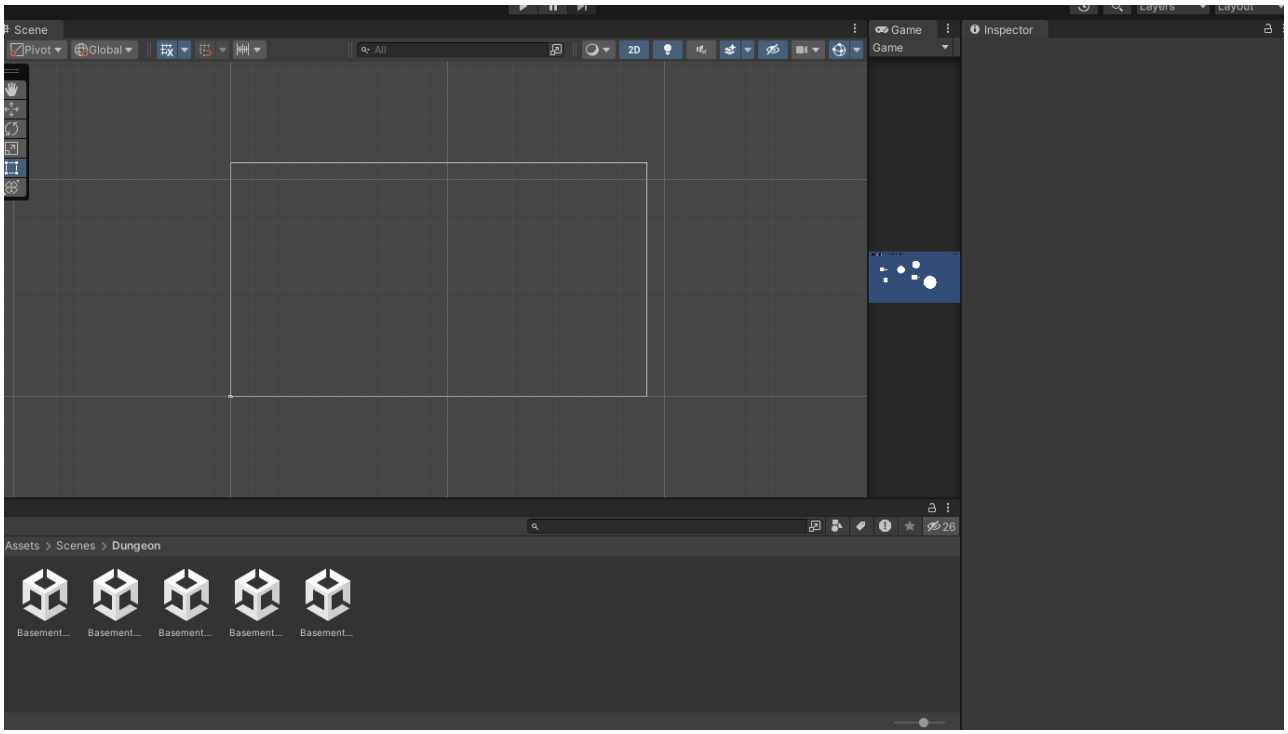


Рисунок 1.20. Стартовий об’єкт в новому проекті Unity

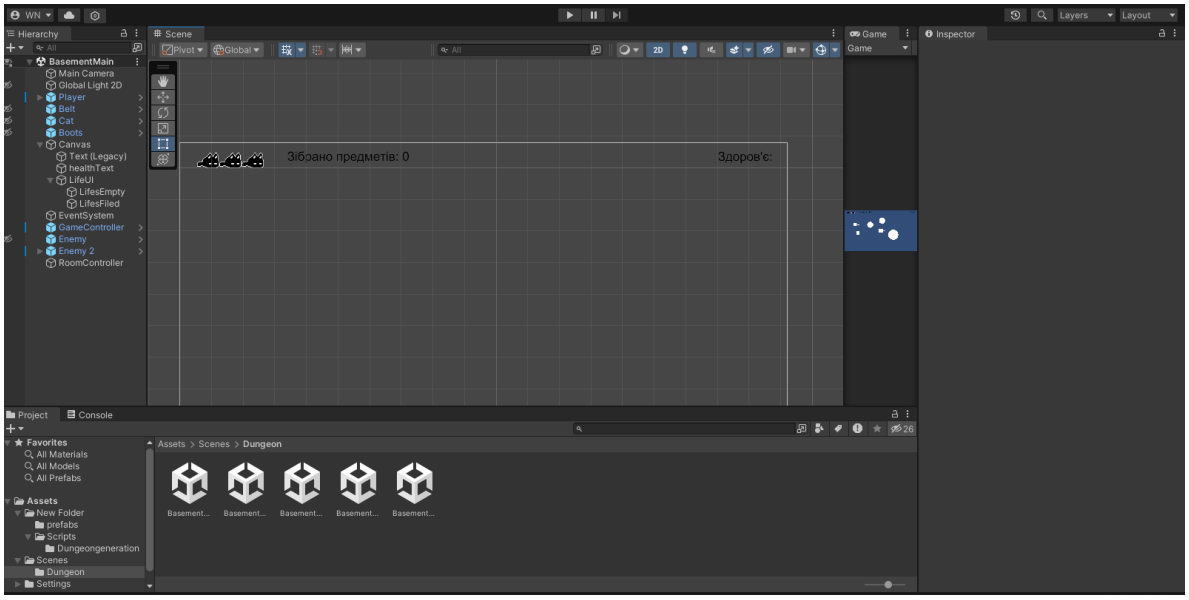


Рисунок 1.21. Весь інтерфейс гри

1.6.1 Кімнати

Наступний крок – створення сцен та додавання об'єктів. Я створив декілька сцен: Для головної кімнати, для початкової кімнати, для пустих кімнат та для кімнати з босом.

Розглянемо кожну кімнату окремо:

В головній кімнаті розташован гравець, камера та “Canvas” для відображення інтерфейсу.

При старті гри, буде підгружатися початкова кімната, де буде знаходитися гравець.

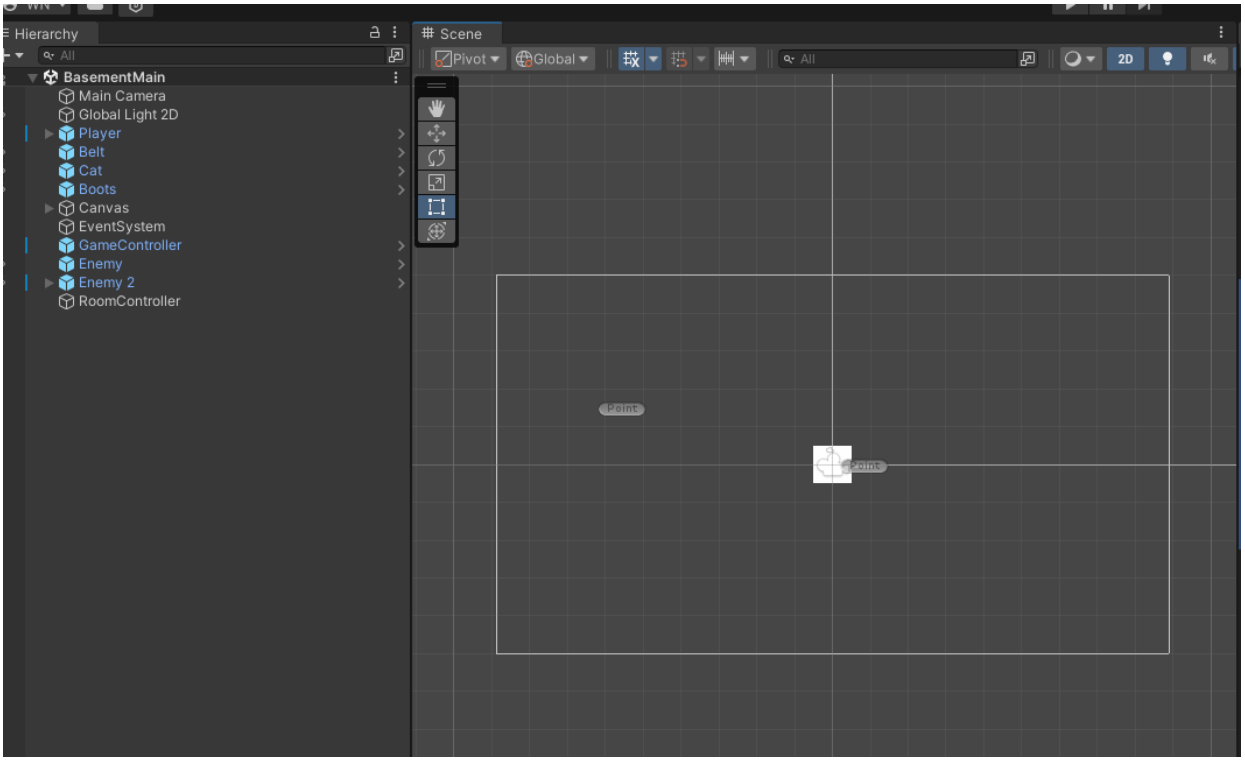


Рисунок 1.22. Головна кімната

В початковій спочатку я додав на сцену підлогу, стіни та двері. На двері та стіни я повісив тег “Ground” для того, щоб при зіштовхуванні пулі зі стіною чи дверима, та видалялась.

Ця кімната буде генеруватися завжди першою. З неї буде починатися весь рівень, та від цієї кімнати будуть генеруватися інщі.

Це буде абсолютно пуста кімната, там буде генеруватися лише гравець і певні предмети. Метою цієї комнати є те, щоб гравець розібрався з початковим управлінням гри.

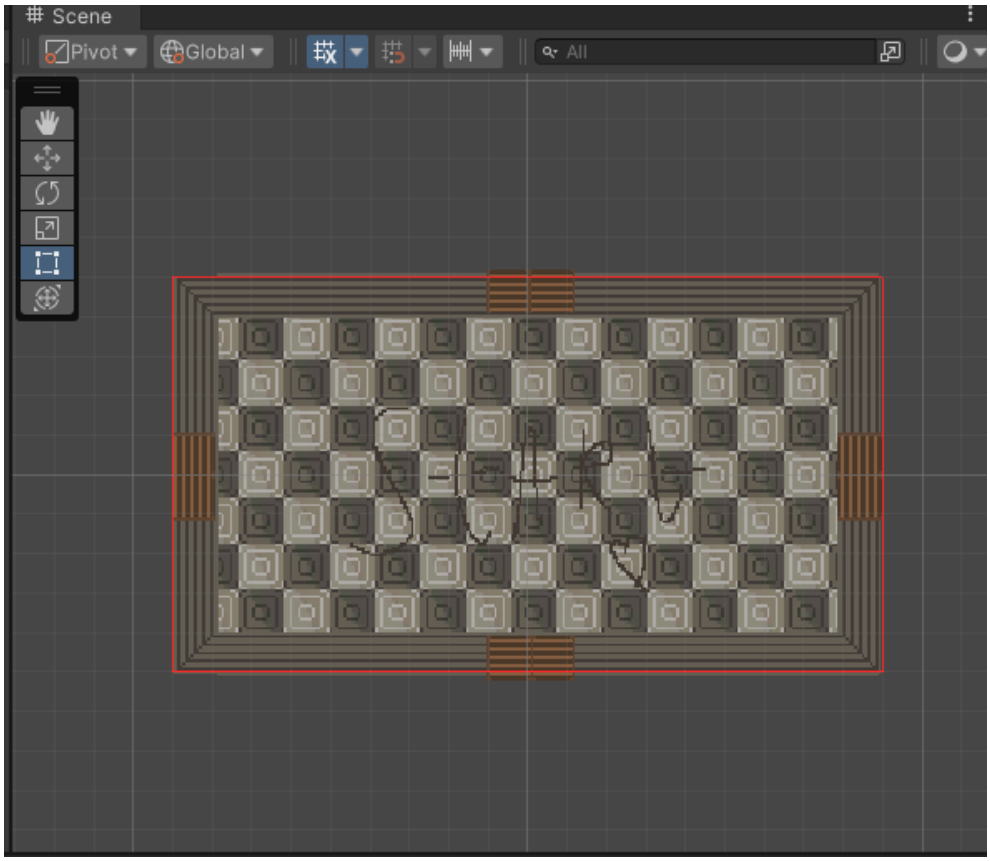


Рисунок 1.23. Початкова кімната

В пуста кімната має все те саме, що і початкова кімната. Вона підгружається згодом для того, щоб із неї генерували подальщі кімнати.

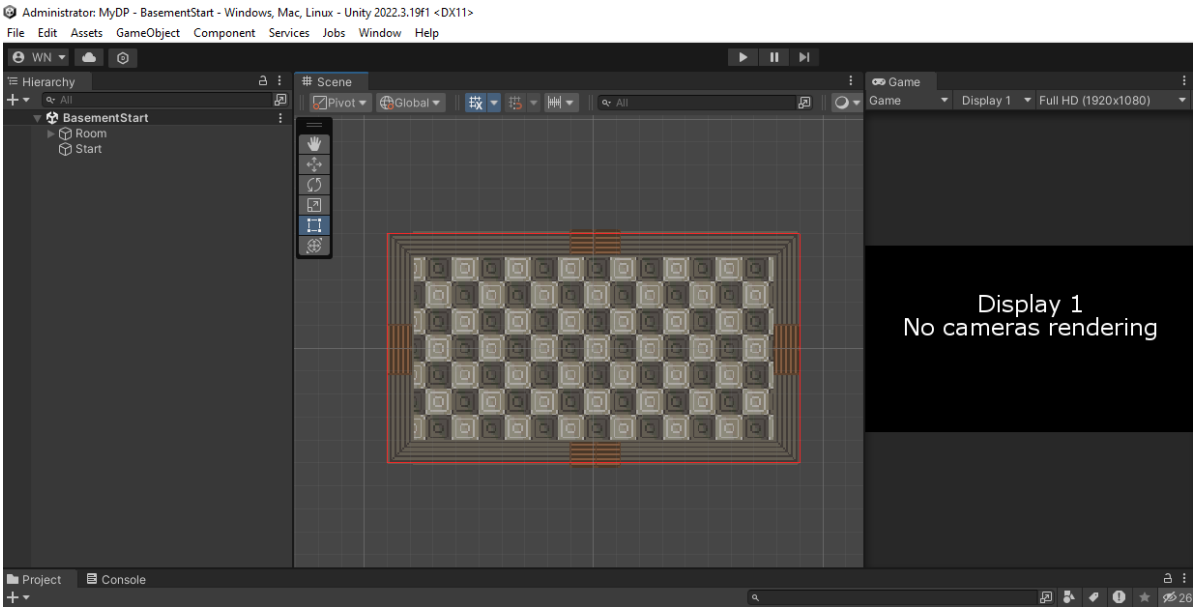


Рисунок 1.24. Пуста кімната

Кімната з босом. Все те саме, що і в пустій, але ще добавляється новий ворог – бос рівня.

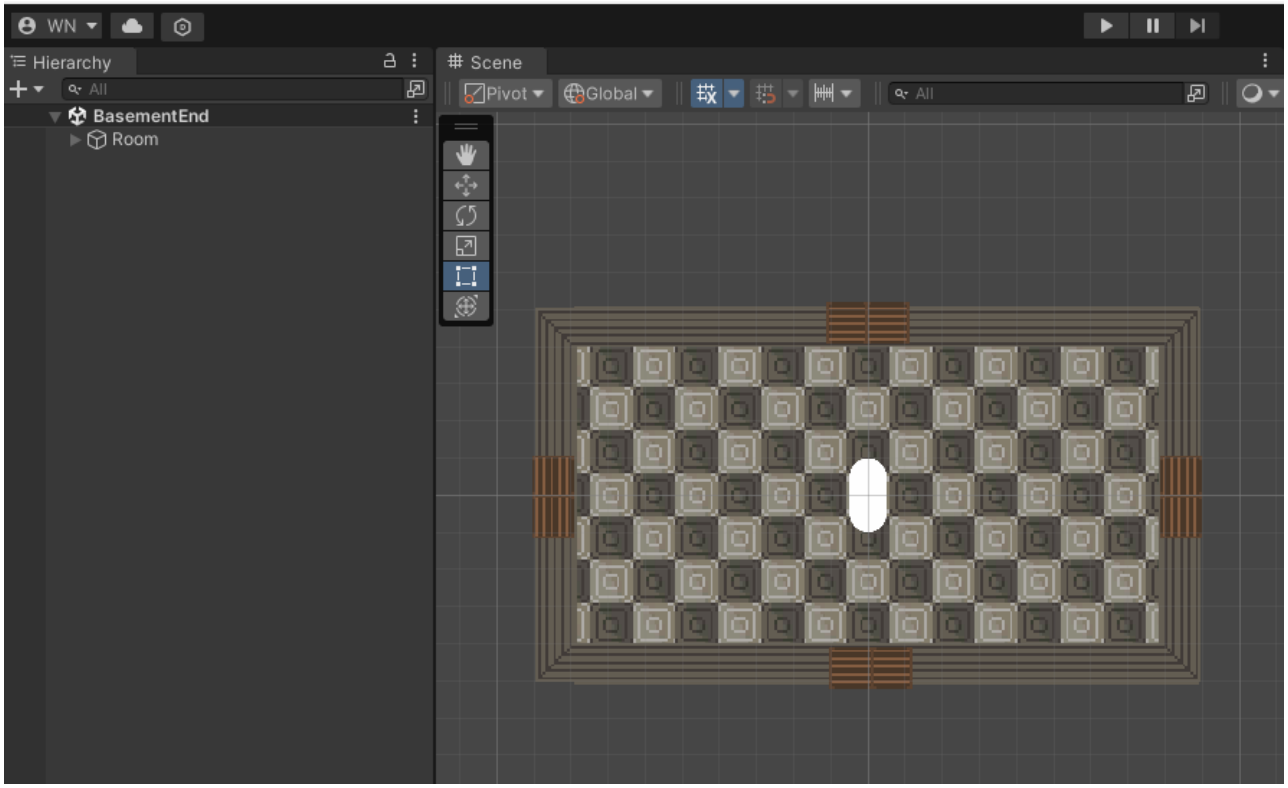


Рисунок 1.25. Кімната з босом

1.6.2 Принцип роботи коду гравця та його стрільба

PlayerController:

Переміщення гравця: Відстежує горизонтальний та вертикальний ввід (клавіші WASD або стрілки) та змінює швидкість об'єкта гравця (rb.velocity).

Відображення зібраних предметів: Оновлює текстовий елемент collectedText, щоб показати кількість зібраних предметів.

Стрільба: Відстежує натискання лівої кнопки миші та викликає метод Shoot для створення кулі (bulletPrefab) і її запуску.

Код скрипту:

```
using UnityEngine;
using UnityEngine.UI;

        public class PlayerController : MonoBehaviour
            {
                public float speed;
                Rigidbody2D rb;
                public Text collectedText;
```

```
        public static int collectedAmount = 0;
        public GameObject bulletPrefab;
        public float bulletSpeed;
        private float lastFire;
        public float fireDelay;

    void Start()
        {
            rb = GetComponent<Rigidbody2D>();
        }
    void Update()
        {
            fireDelay = GameController.FireRate;
            speed = GameController.MoveSpeed;
            fireDelay = GameController.FireRate;
            speed = GameController.MoveSpeed;
            float horizontal = Input.GetAxis("Horizontal");
            float vertical = Input.GetAxis("Vertical");
            rb.velocity = new Vector3 (horizontal * speed, vertical *
        speed, 0);
            collectedText.text = "Item collected: " + collectedAmount;
        }
    }
```

Цей скрипт PlayerController в Unity відповідає за управління гравцем, а саме: його рух та відображення зібраних предметів.

Код PlayerController в Unity відповідає за управління рухом гравця та відображення кількості зібраних предметів на екрані. Він використовує компонент Rigidbody2D для обробки фізики руху гравця. У методі Start ініціалізується цей компонент.

У методі Update, який викликається кожен кадр, зчитуються дані про швидкість руху та затримку пострілу з GameController. Потім обробляється введення з клавіатури гравцем, визначаючи напрямок руху гравця. Швидкість гравця оновлюється відповідно до введення з клавіатури гравцем і заданої швидкості. Також у кожному кадрі оновлюється текстовий елемент на екрані, що показує кількість зібраних предметів.

Таким чином, цей скрипт дозволяє рухатися гравцеві по ігровій сцені та відображати кількість зібраних предметів в інтерфейсі.

Weapon:

Поворот зброї: Обчислює кут повороту на основі позиції миші відносно об'єкта зброї та встановлює відповідний поворот.

Стрільба: Відстежує час затримки між пострілами та натискання лівої кнопки миші. Якщо час затримки минув, створює кулю (bullet) в позиції точки стрільби (point) і встановлює її поворот.

Код скрипту:

```
using UnityEngine;
	public class Weapon : MonoBehaviour
		{
			public float offset;
			private float time;
			public float startTime;
			public GameObject bullet;
			public Transform point;
			void Update()
		{
			Vector3 mousePosition =
			Camera.main.ScreenToWorldPoint(Input.mousePosition);
			Vector3 difference = mousePosition - transform.position;
			difference.z = 0f;
			float rotateZ = Mathf.Atan2(difference.y, difference.x) *
			Mathf.Rad2Deg;
			transform.rotation = Quaternion.Euler(0f, 0f, rotateZ + offset);
		}
			if (time <= 0f)
				{
				if (Input.GetMouseButtonDown(0))
					{
						Instantiate(bullet, point.position,
		point.rotation);
						time = startTime;
					}
				}
			else
				{
					time -= Time.deltaTime;
				}
			}
		}
```

Скрипт Weapon в Unity відповідає за управління зброєю персонажа. Він обробляє обертання зброї за напрямком курсора миші, а також його стрільбу. У

методі Update кожен кадр обчислюється положення курсора миші в глобальних координатах, потім розраховується різниця між положенням миші і зброї, і на основі цього обчислюється кут повороту зброї гравця. Якщо гравець натискає ліву кнопку миші і затримка між пострілами пройшла, створюється куля в положенні зброї, і таймер затримки перезапускається. Також зброя була прикріплена до гравця та стала дочернім.

Нижче приведені скриншоти вигляду гравця і налаштування зброї.

Рисунок 1.26.Структура гравця

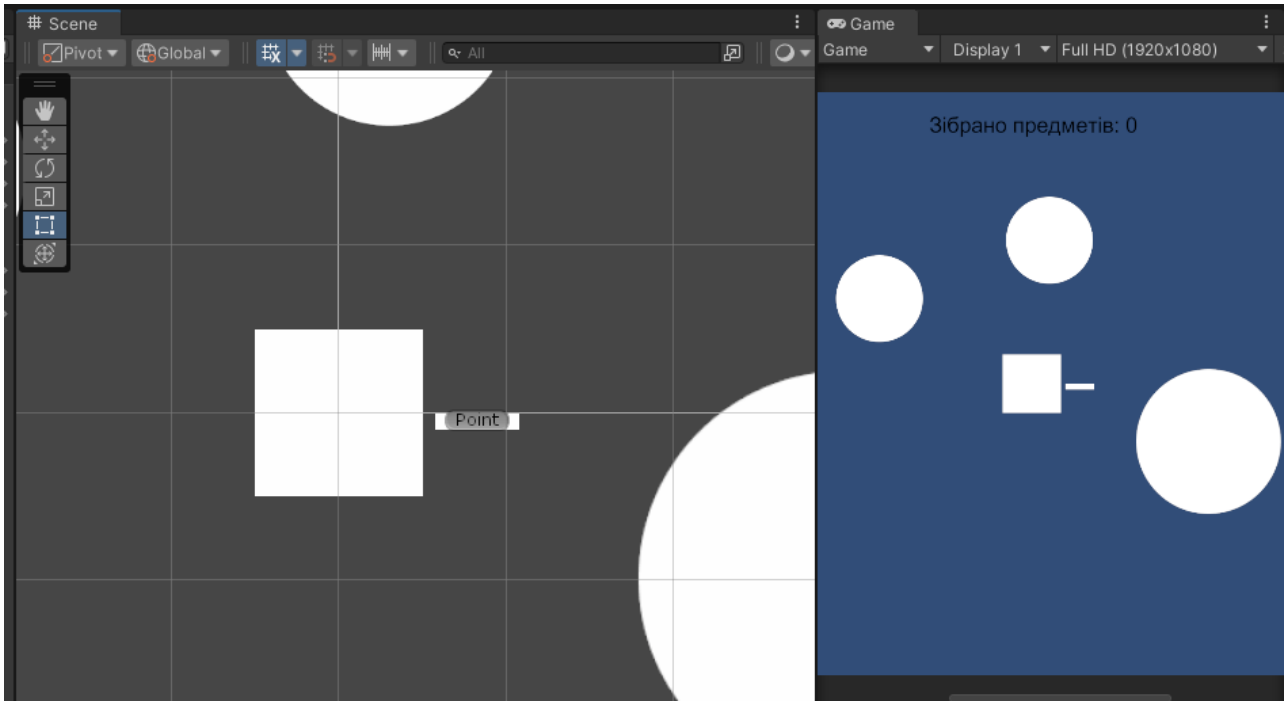


Рисунок 1.27. Вигляд гравця та зброї

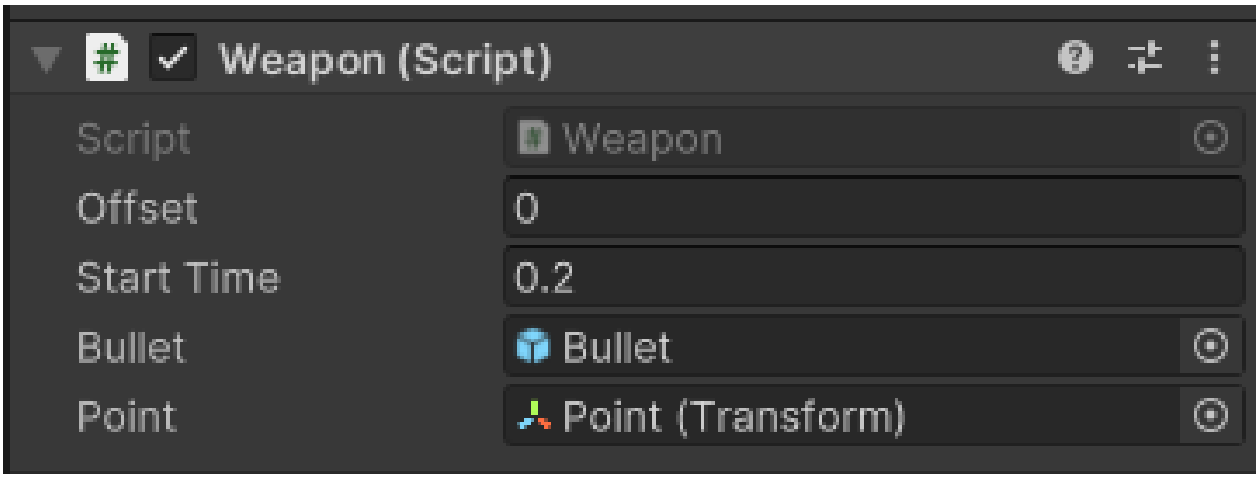


Рисунок 1.28. Налаштування зброї

1.6.3 Камера гравця

Скрипт CameraController в Unity відповідає за керування камерою, котра слідує за гравцем, коли той переміщується між кімнатами. У методі Update кожен кадр викликається UpdatePosition, що переміщує камеру до центру поточної кімнати. Якщо кімната не задана, метод повертає камеру в початкове положення. Метод IsSwitchingScene визначає, чи камера знаходиться в процесі переміщення між кімнатами. Клас також має статичний екземпляр instance, що дозволяє іншим скриптам легко звертатися до нього.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class CameraController : MonoBehaviour
        {
                public static CameraController instance;
                public Room currRoom;
                public float moveSpeedWhenRoomChange;
                void Awake()
        {
                instance = this;
        }
                // Update is called once per frame
                void Update()
        {
                UpdatePosition();
        }
                void UpdatePosition()
        {
        if (currRoom == null)
```

```
        {
            return;
        }
            Vector3 targetPos = GetCameraTargetPosition();
             transform.position =
            Vector3.MoveTowards(transform.position, targetPos,
            Time.deltaTime * moveSpeedWhenRoomChange);
        }
            Vector3 GetCameraTargetPosition()
        {
        if (currRoom == null)
        {
            return Vector3.zero;
        }
            Vector3 targetPos = currRoom.GetRoomCentre();
            targetPos.z = transform.position.z;
            return targetPos;
        }
        public bool IsSwitchingScene()
        {
         return transform.position.Equals(GetCameraTargetPosition()) ==
         false;
        }
    }
```

Скрипт Enemy визначає поведінку ворогів у грі. Ворог має здоров'я, і якщо воно опускається до нуля, об'єкт знищується. Ворог може перебувати в чотирьох станах: блукання (Wander), слідування за гравцем (Follow), атака (Attack) і смерть (Die). У стані блукання ворог випадковим чином вибирає напрямок руху. У стані слідування ворог рухається до гравця, якщо той перебуває в межах заданого радіуса. У стані атаки ворог або завдає шкоди в ближньому бою, або стріляє по гравцеві залежно від типу ворога (Melee або Ranged). При цьому використовуються тимчасові затримки для реалізації атак. Якщо гравець опиняється в радіусі атаки, стан ворога перемикається на атаку.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public enum EnemyState
    {
        Wander,
        Follow,
        Die,
```

```
        Attack
};
public enum EnemyType
{
        Melee,
        Ranged
};
public class EnemyControle : MonoBehaviour
{
    GameObject player;
    public EnemyState currState = EnemyState.Wander;
    public EnemyType enemyType;
    public float range;
    public float speed;
    public float attackRange;
    public float coolDown;
    private bool chooseDir = false;
    private bool dead = false;
    private bool coolDownAttack = false;
    private Vector3 randomDir;
    public GameObject Bullet;
    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player");
    }
    // Update is called once per frame
    void Update()
    {
        switch (currState)
        {
            case (EnemyState.Wander):
                Wander();
            break;
            case (EnemyState.Follow):
                Follow();
            break;
            case (EnemyState.Die):
            break;
            case (EnemyState.Attack):
                Attack();
            break;
        }
        if(IsPlayerInRange(range) && currState != EnemyState.Die)
        {
            currState = EnemyState.Follow;
        }
```

```
        else if(!IsPlayerInRange(range) && currState != EnemyState.Die)
        {
            currState = EnemyState.Wander;
        }
        if (Vector3.Distance(transform.position, player.transform.position)
<= attackRange)
        {
            currState = EnemyState.Attack;
        }
    }
    private bool IsPlayerInRange(float range)
    {
        return Vector3.Distance(transform.position,
player.transform.position) < range;
    }
    private IEnumerator ChooseDirection()
    {
        chooseDir = true;
        yield return new WaitForSeconds(Random.Range(2f, 8f));
        randomDir = new Vector3(0, 0, Random.Range(0, 360));
        Quaternion nextRotation = Quaternion.Euler(randomDir);
        transform.rotation = Quaternion.Lerp(transform.rotation,
nextRotation, Random.Range(0.5f, 2.5f));
        chooseDir = false;
    }
    void Wander()
    {
        if(!chooseDir)
        {
            StartCoroutine(ChooseDirection());
        }
        transform.position += -transform.right * speed * Time.deltaTime;
        if (IsPlayerInRange(range))
        {
            currState = EnemyState.Follow;
        }
    }
    void Follow()
    {
        transform.position = Vector2.MoveTowards(transform.position,
player.transform.position, speed * Time.deltaTime);
    }
    void Attack()
    {
        if (!coolDownAttack)
        {
```

```
            switch (enemyType)
            {
                case (EnemyType.Melee):
                    GameController.DamagePlayer(1);
                    StartCoroutine(CoolDown());
                    break;
                case (EnemyType.Ranged):
                    break;
            }
        }
    }
    private IEnumerator CoolDown()
    {
        coolDownAttack = true;
        yield return new WaitForSeconds(coolDown);
        coolDownAttack = false;
    }

    public void Death()
    {
        Destroy(gameObject);
    }
}
```

1.7 Тестування працездатності ігрових елементів

Тестування є невід'ємною частиною процесу розробки програмного забезпечення та інших технічних продуктів. Воно призначене для перевірки якості продукту, виявлення помилок та забезпечення його надійності. Тестування допомагає гарантувати, що кінцевий продукт відповідає вимогам і очікуванням користувачів.

Основною метою тестування є виявлення помилок та дефектів у програмному забезпеченні до його випуску на ринок. Тестування забезпечує якість продукції, перевіряючи відповідність технічним вимогам і специфікаціям. Воно включає в себе перевірку функціональних можливостей, продуктивності, безпеки, сумісності та простоти використання. Високоякісне програмне забезпечення підвищує задоволеність користувачів та зменшує ймовірність виникнення проблем під час експлуатації. При випуску продукту з помилками може статись втрата фінансів, включаючи витрати на виправлення помилок після випуску та втрату клієнтів. Користувачі очікують, що програмне

забезпечення буде працювати без збоїв та відповідати їхнім потребам. Тестування допомагає забезпечити, що продукт відповідатиме їх очікуванням, надаючи їм позитивний досвід використання. Це, в свою чергу, сприяє утриманню клієнтів та підвищенню лояльності. Також тестування дозволяє переконатися, що продукт відповідає вимогам та специфікаціям, визначеним на етапі планування та розробки. Це включає функціональні вимоги, вимоги до продуктивності, безпеки та сумісності. Відповідність вимогам гарантує, що продукт буде працювати належним чином у реальних умовах.

Тестування продуктивності дозволяє оцінити, як програмне забезпечення працює під навантаженням та чи здатне воно масштабуватися відповідно до зростаючих вимог. Це важливо для забезпечення безперебійної роботи продукту в умовах високої активності користувачів. Таким чином, тестування є критично важливим етапом розробки програмного забезпечення, що забезпечує його якість, надійність та відповідність вимогам. Воно допомагає зменшити ризики, підвищити задоволеність користувачів та запобігти фінансовим втратам, що робить його невід'ємною частиною успішного впровадження програмних продуктів.

Завдяки можливостям ігрового рушія Unity. тестування гри можна проводити на різних етапах розробки. Для цього достатньо розмістити ігрові об’єкти на свої місця на сцені, виставити всі необхідні ігрові параметри та усунути проблеми. Це важливо для коректної роботи програми та ігрового досвіду гравця. Після виконання перших налаштувань, необхідно натиснути на кнопку «Play». Після чого гра запуститься.

Рисунок 1.29. Приклад тестування гри

Перша проблема, з якою я зустрівся – це неправильне розміщення кімнати боса. Вона генерувалася поверх основної кімнати, що ламало всю структуру рівня. Але рішення стало дуже простим: треба було просто поміняти в скрипті появи стартової кімнати іншу сцену, бо через це підгружалась зовсім не та сцена. І треба було усунути дублювання, щоб появлялась лише одна стартова кімната на рівні.

Рисунок 1.29. Приклад багу

Основною метою тестування є перевірка справності всіх механік гри, а саме:

Переміщення гравця

Переміщення противника

Нанесення пошкоджень

Відображення інтерфейсу

Генерація рівня

Стрільба

Генерація предметів

Генерація ворогів

Причини для такого тестування лежать у фізичній системі переміщення гравця та ворогів і необхідно переконатись, що їх взаємодія не приводить до некоректних ситуацій. Тестування показало справну роботу фізики, вороги коректним чином реагують на зіткнення з гравцем, стінами , наносить йому пошкодження.

Наступним етапом тестування є перевірка генерації кімнат на рівні. Для цього достатньо запустити гру і подивитися на сцену.

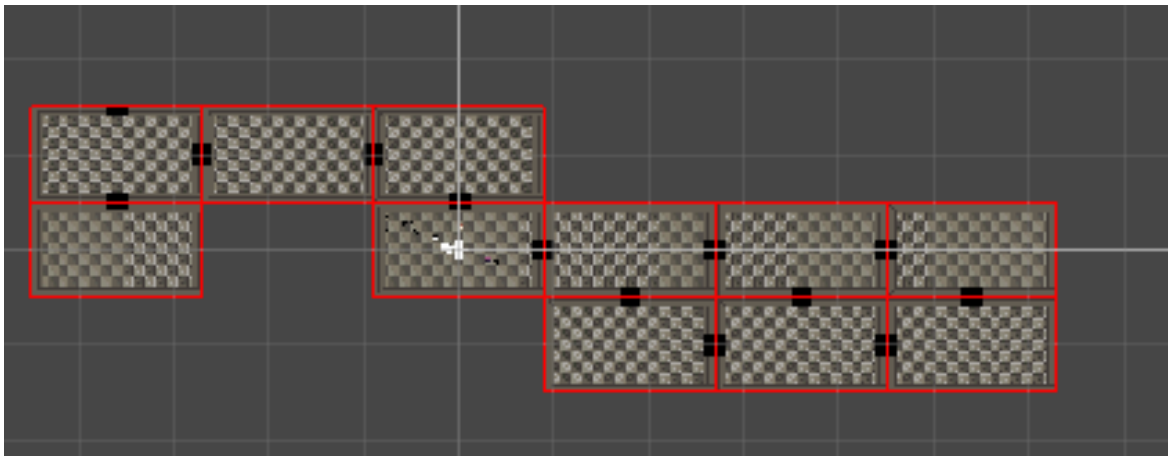


Рисунок 1.29. Приклад Генерації кімнат

Далі було проведено тестування пострілів гравця та ворогів. Це одна з важливих частин перевірки, бо це одна з головних механік гри.

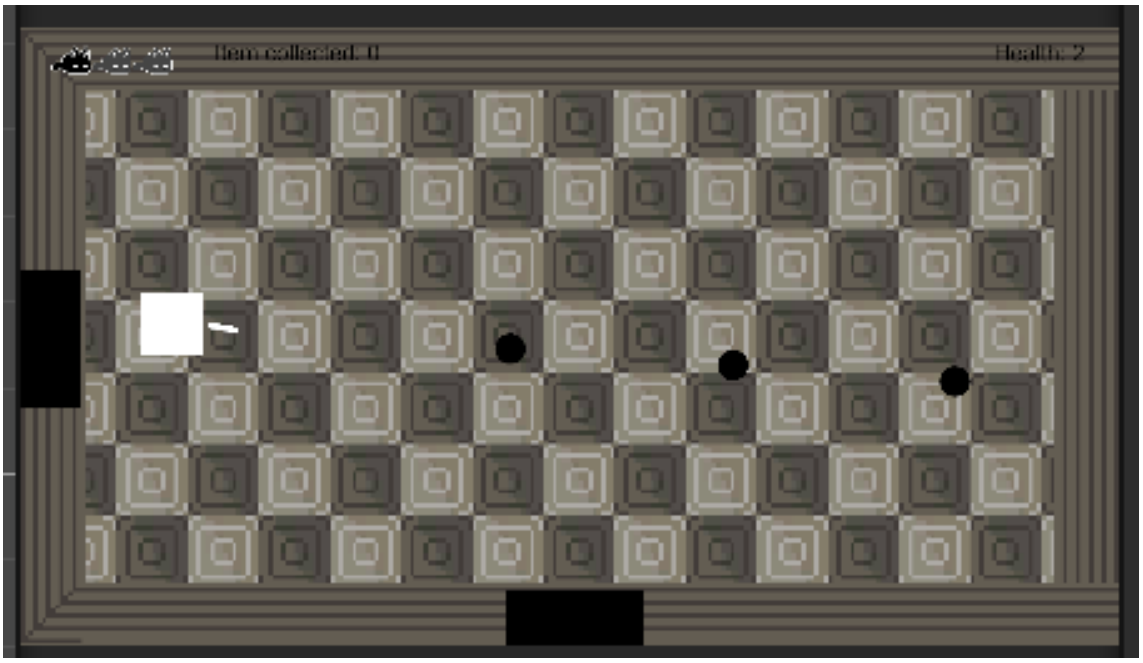


Рисунок 1.30. Приклад стрільби гравця

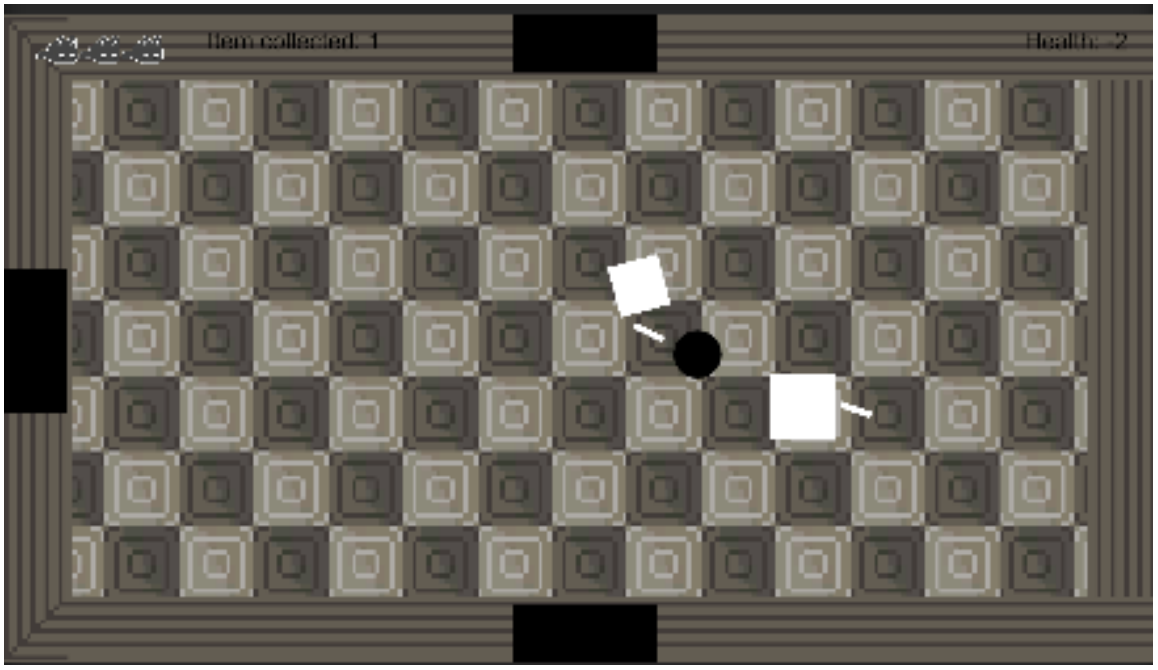


Рисунок 1.31. Приклад стрільби противника

Виконавши всі ці етапи тестування, можна переконатися, що гра коректно виконує основні вимоги, сформовані під час проектування проекту. Подальші модифікації гри можуть розширити ігровий процес та покращити його якість.

Таким чином, процес тестування є надзвичайно важливим для створення якісного, надійного та задовільного для користувачів програмного забезпечення. Він допомагає не тільки виявити та виправити помилки до випуску продукту, але й забезпечує відповідність програмного забезпечення технічним вимогам та специфікаціям. Завдяки ретельному тестуванню можна запобігти фінансовим втратам, підвищити задоволеність користувачів та зміцнити їхню довіру до продукту. Тестування дозволяє виявити та вирішити проблеми на ранніх етапах розробки, що робить процес розробки більш ефективним та економічним. Успішне тестування є запорукою успішного впровадження та подальшого розвитку програмного забезпечення.

2 Економічний розділ

2.1 Резюме

В рамках цього дипломного проекту була розроблена рання версія 2D-гри жанру роуглайк, а також повністю завершена робота над програмною частиною базового ігрового інтерфейсу, характерного для даного жанру ігор, і взаємодією з ігровим процесом. Використовуючи принципи модульної розробки, я зміг внести цікаві аспекти в гру, додавши ігрові елементи, механізми або режими.

Розробка відеоігор є комплексним процесом, що вимагає значних фінансових інвестицій. Для успішного втілення проекту необхідно враховувати різноманітні економічні аспекти, включаючи витрати на розробку, маркетинг, а також прогнозовані доходи.

2.2 Визначення трудомісткості розробки програмного забезпечення

Визначення трудомісткості розробки програмного забезпечення є критичним етапом планування проекту, що дозволяє оцінити необхідні ресурси, строки виконання завдань та загальні витрати. Тривалість розробки програмного продукту залежить від його кількості, складності розробки і планованого терміну, який визначається ринковими умовами.

У таблиці 2.1 наведені аналоги програмного забезпечення, і їх функції в більшій чи меншій мірі виконуються розробленими програмними продуктами. У нашій версії це виділено сірим.

Таблиця 2.1 Каталог аналогів програмного забезпечення

Найменування ПП	*Обсяг функції ПП – V_o, усл. машинних командах.*
1. ПП СУБД	2500 - 9800
2. Комплексні системи ведення БД	950 - 7430
3. ПП загальної математики і ПП імітаційного моделювання	7800 - 8800

При виборі аналога ПП з Vo в умовній машинної команді складність визначається на підставі таблиці 2.2

Обсяг ПП, тис.умов.машинних команд	Норма часу, люд/год
1.00	229
2.00	244
3.00	262

На підставі отриманих значень, згідно з довідником, встановлюється розширена норма часу на розробку програмних аналогів (коригується з поправочним коефіцієнтом з урахуванням умов розробки програмного забезпечення, тобто Кк =0,7÷0,8 в комп'ютерних умовах): Тар=229х0,7= 160,3 (люд/год).

Трудомісткість програмного продукту заснована на складності аналога з урахуванням складності розробки, ступеня новизни, ступеня використання при розробці стандартних модулів на основі формул:

$$T_{ТЗ} = T^a p \times L_1 \times K_H \quad (2.1)$$

$$T_{ТП} = T^a p \times L_2 \times K_H \quad (2.2)$$

$$T_{РП} = T^a p \times L_3 \times K_H \times K_T \quad (2.3)$$

Для розрахунку потрібні наступні коефіцієнти:

Li - питома вага на етапі розробки збірок (Таблиця 2.3); Кн - поправочний коефіцієнт з урахуванням ступеня новизни (таблиця 2.4); Кт- поправочний коефіцієнт з урахуванням ступеня його використання при розробці типових програм (таблиця 2.5).

Таблиця 2.3. Значення питомих коефіцієнтів трудомісткості стадії в загальній трудомісткості розробки ПП.

Код стадії	Ступінь новизни		
	А	Б	В
ТЗ (L_1)	0,15	0,12	0,12
ТП (L_2)	0,16	0,15	0,11
РП (L_3)	0,55	0,58	0,61

У нашій версії він виділений сірим кольором.

Таблиця 2.4. Значення поправочного коефіцієнта з урахуванням ступеня новизни

Код ступеня новизни	Ступінь новизни	Значення $К_н$
А	Принципово нові ПП	1,75 – 1,2
Б	ПП – розвиток визначеного параметричного ряду	1,0 – 0,8
В	ПП маючий аналог	0,7

У нашій версії він виділений сірим кольором.

Таблиця 2.5. Значення коефіцієнта ступеня використання при розробці стандартних програм

Ступінь охоплення реалізованих функцій розроблювального ПП типовими програмами, %	Значення $К_т$
60 і вище	0,6
40-60	0,7
20-40	0,8
До 20	0,9

У нашій версії він виділений сірим кольором.

Тепер розрахуйте трудомісткість для кожного етапу окремо:

Трудомісткість технічного завдання:

$Т_{тз}=Та*L_1*К_н=160,3*0,12*0,7=13,46$(люд/годин)

Трудомісткість розробки технічного проекту:

$Т_{тп}=Та*L_2*К_н=160,3*0,11*0,7=12,34$ (люд/годин)

Трудомісткість розробки робочого проекту:

$Т_{рп}=Та*L_3*К_н*К_т=160,3*0,61*0,7*0,8=54,75$ (люд/годин)

Для подальших розрахунків визначили кількість папера, витраченого на кожен етап: технічне завдання $N_{тз}=2$ (стр), розробка ТП $N_{тп}=40$ (стр), розробка робочого проекту $N_{рп}=23$ (стр), пояснювальна записка відповідно $N_{пз}=20$ (стр). Розрахунок зведений у таблицю 2.6.

Таблиця 2.6. Розрахунок трудомісткості ПП

Найменування етапів	Розрахунок, годин.		
1	2	3	4
1.ТЗ	$T_{Ртз}$=13,46	$T_{кк}=0,7*N_{тз}$= 0,7*2=1,4	$T_{нк}=0,15*N_{тз}$=0,15*2=0,30
2.Розробка ТП	$T_{Ртп}$=12,34	$T_{кк}=0,7*N_{тп}$=0,7*40= 28	$T_{нк}=0,15*N_{тп}$=0,15*28=4,2
3.Розробка РП	$T_{Ррп}$= 54,75	$T_{кк}=0,7*N_{рп}$=0,7*23= =16,1	$T_{нк}=0,15*N_{рп}$=0,15*14=2,4
4.Розробка ПЗ	$T_{пз}=1,5*N_{пз}$ = 1,5*20 =30	$T_{кк}=0,7*N_{тз}$=0,7*20 =14	$T_{нк}=0,15*N_{пз}$=0,15*20 =3
Усього, в т.ч.:	179,95		
- на розробку	$\Sigma T_р$=110,55		
- контроль керівника		$\Sigma T_{кк}$=59,5	
- нормконтроль			$\Sigma T_{нк}$=9,9

2.3 Розрахунок ціни програмного продукту

Визначення ціни програмного продукту є ключовим етапом у процесі його комерціалізації. Розрахунок базової заробітної плати виконавця наведено в таблиці 2.7. Стаття 8 Закону Про державний бюджет України на 2024 рік. Мінімальна щомісячна заробітна плата встановлена на рівні 8000 гривень в період з 2024; мінімальна погодинна тарифна ставка становить 48 грн.

Таблиця 2.7. Розрахунок основної заробітної плати виконавців.

Найменування робіт	Трудомісткість робіт, години	Погодинна тарифна ставка, грн.	Розрахунок, грн.
1.Розробка ПП	110,55	48	5306,4
2.Контроль керівника	59,5	68,10	4051,95
3.Нормоконтроль	9,9	68,10	674,19
Усього	-	-	ΣЗо= 10032,54

Розрахувати матеріальні витрати на розробку ПП. Розрахунки наведені в таблиці 2.8:

Таблиця 2.8. Розрахунок матеріальних витрат на розробку ПП

Найменування матеріальних витрат	Тип, модель	Кількість	Ціна одиниці, грн.	Вартість, грн.
Папір	Лист А4	87	3,50	104,5
Разом	-	-	-	$В_{м\,і}$=104,5
Транспортно – заготівельні Витрати (10%)				$Втр\_з = 0{,}1 \times Вм1$= 0,1*104,5=10,45
Усього				$В_м$=$В_{мі}$+$В_{тр\_з=}$ 114,95

На підставі даних, отриманих за окремими статтями витрат, розрахунок планових витрат за ПП в цілому узагальнено у форматі, наведеному в таблиці 2.9.

Таблиця 2.9. Розрахунок статей витрат планової собівартості

Стаття витрат	Значення, грн.	Формула розрахунку
1. Матеріали	114,95	$В_м$ (див. табл. 2.8)
2. Основна заробітна плата	10032,54	$З_о$ (див. табл. 2.7.)
3.Додаткова заробітна плата	1003,25	$Зд = 0{,}1 \times Зо$ = 10032,54*0,1
4.Відрахування до єдиного фонду соціального внеску	2427,87	$Вє.с.в. = 0{,}22 \times (Зо + Зд)$=0,22* (10032,54+1003,25)
5. Накладні витрати	4012,8	$Внак. = 0{,}4 \times Зо$ =0,4 * 10032,54
6. Повна собівартість	17591,41	$С_{пов}$= $В_м$ +$З_о$ + $З_д$ +Вє.с.в. + Внак.= 114,95+10032,54+1003,25+2427,87+4012,8

Розмір включеної в ціну прибутку визначається за такою формулою:

$$П=(С_п*р)/100=(17591,41*10)/100=1759,14 \text{ грн}$$

Де р – плановий рівень рентабельності (10-15%).

Оптова ціна (кошторисна вартість) визначається по формулі:

$$Ц_о = С_п + П = 17591,41+1759,14= 19350,55 \text{ грн};$$

Податок на додану вартість визначаємо по наступній формулі:

$$ПДВ = 0,2 * Цо = 0,2 *19350,55 = 3870,11 \text{ грн};$$

Виходячи з отриманих даних, ціна продажу продукту, буде:

$$Ц_р = Ц_о + ПДВ = 19350,55 + 3870,11 = 23220,66 \text{ грн};$$

3 Розділ охорони праці та техніки безпеки

3.1 Вступ

Охорона праці є важливим чинником на підприємстві. Вона відіграє важливу роль у безпечному перебуванню працівників на їх робочому місці, бо наскільки б не були важливими здобутки працівників, вони ніколи не зможуть повернути втраченого здоров’я, або життя.

Програмісти, які займаються цим ремеслом, проводять багато часу за комп'ютером, що може негативно впливати на їхнє здоров'я та працездатність. Також така робота спричиняє значну розумову та нервово-емоційну напругу, навантаження на зоровий апарат і м’язів кістки зап’ястка, п’ястка та фаланги пальців. Важливо мати конструкцію для роботи, яка розташовує всі елементи робочого місця так, щоб робоча поза людини-оператора не була викривлена чи приносила дискомфорт. Тому важливо створювати оптимальні умови праці для програмістів, аби мінімізувати ризики, пов'язані з їхньою професією, а згідно зі статтею 13 Закону України «Про охорону праці» роботодавець зобов’язаний створити на кожному робочому місці у всіх структурних підрозділах умови праці відповідно до нормативно-правових актів, також забезпечити додержання і підтримання вимог законодавства щодо прав працівників.

Цей розділ присвячено вивченню оптимальних умов праці програміста, а також його обов'язків з охорони праці при розробці комп'ютерної 2D-гри у жанрі роуглайк на програмному рушії Unity.

3.2 Аналіз небезпечних, шкідливих чинників для працівника

Оператор ПК на робочому місці зіштовхується з купою чинників, які можуть призвести до погіршення його працездатності. Загалом їх можна поділити на 4 типи: фізичні, психофізіологічні, санітарно-гігієнічні та соціально-психологічні. До фізичних чинників ми можемо віднести неправильну ергономіку робочого місця, високий рівень шуму у приміщенні, вібрації.

3.2.1 Пожежна безпека

Приміщення з ЕОМ повинні бути оснащені системою автоматичної пожежної сигналізації відповідно до вимог Переліку однотипних за призначенням об'єктів, які підлягають обладнанню автоматичними установками пожежогасіння та пожежної сигналізації, з димовими пожежними сповіщувачами та переносними вуглекислотними вогнегасниками з розрахунку 2 шт. на кожні 20 кв. м площі приміщення з урахуванням граничнодопустимих концентрацій вогнегасної рідини відповідно до вимог Правил пожежної безпеки в Україні. В інших приміщеннях допускається встановлювати теплові пожежні сповіщувачі.

Основними причинами виникнення пожежі в кабінеті інженера-програміста можуть бути:

- - замикання або загоряння електрообладнання, що експлуатується (монітор, принтер, клавіатура і т.д.);
- - додаткові опалювальні прилади;
- - система штучного освітлення;
- - загоряння паперових документів.

3.3 Вимоги до виробничого середовища

Гігієнічні вимоги до виробничого середовища ґрунтуються на принципах запобігання негативному впливу факторів виробничого середовища на здоров'я та працездатність людини. Їх дотримання є запорукою створення безпечних та комфортних умов праці, що сприяє підвищенню продуктивності та якості роботи.

3.3.1 Вимоги до приміщення

Об’ємно-планувальні характеристики приміщень та будівель, де використовуються візуальні дисплейні термінали (ВДТ), повинні відповідати вимогам ДСанПІН 3.3.2.007-98. Також приміщення з вмістом візуальних дисплейних терміналах (ВДТ), електронно-обчислювальних машин (ЕОМ) та персональних електронно-обчислювальними машинами (ПЕОМ) заборонено розміщювати у підвальних, або цокольних поверхах. Площа приміщення має

становити не менше 6 м^2, а об’єм не менше ніж 20,0 м^3 на одне робоче місце. Кожен день має проводитись вологе прибирання. Самі ж робочі місця повинні бути розташовані не менше ніж на відстані 1 м від стіни з вікном, і 1,4 м від звичайної стіни. Між бічними поверхнями комп’ютерів відстань має бути не меншою за 1,2 м, а між тильною поверхнею комп’ютера та екраном іншого не повинна бути не меншою 2,5 м. Приміщення має бути обладнене меблями для зберігання документів за урахування умов до площі. Також забороняється до використання полімерних матеріалів (деревинно-стружкові плити, шпалери, що миються, рулонні синтетичні матеріали, шаруватий паперовий пластик тощо), бо вони виділяються шкідливі хім. речовини у повітря. Покриття підлоги ж має бути матовим, поверхня – рівною, не слизькою, з властивостями антистатика. Для стін пастельні кольори використовують, як основні.

3.3.2 Освітлення

Особливу увагу треба приділити освітленню так як, саме воно впливає не тільки на збереження здоров’я, а і на якість праці та продуктивність працівників. Відповідно до ДБН В.2.5-28:2018 «Природне і штучне освітлення» приміщення має мати, що природнє, що штучне освітлення, що рівномірно розподілено. Природне освітлення має забезпечувати коефіцієнт природної освітленості (КПО) не нижче ніж 1,5%. Для регулювання рівня освітлення природним світлом бажано застосовувати жалюзі. Також обладнене робоче місце має бути розміщено так, щоб уникнути попадання прямого сонячного світла у очі. Застосування світильників екрануючих сіток, або без розсіювачів забороняється. В свою чергу рівень освітленості на робочому столі та в зоні зберігання документів має становити 300-500 лк.

3.3.3 Шум

Рівень шуму, вібрацій в робочих приміщеннях, що працюють з ПК визначаються відповідно до ДСанПІН 3.3.2.007-98. На робочих місцях працівників нормуються рівні звуку(для програмістів він становить – 50 дБА). Їх вибір залежить від інденеро-акустичних розрахунків(п. 3.3.3 ДСанПіН 3.3.2.007-98).

3.3.4 Мікроклімат

У виробничих приміщеннях на робочих місцях мають забезпечуватись оптимальні значення параметрів мікроклімату, а сме: температура, відносна вологість та рухливість повітря за ДСН 3.3.6.042-99 «і норми мікроклімату виробничих приміщень». Робочі приміщення мають бути обладнені системою опалення, кондиціонування повітря, або припливно-витяжною вентиляцією. Також кількість теплоти, що генерується системою опалення, иає бути відповідною до втрат теплоти у приміщенні.

Параметри мікроклімату	Значення	Параметри
	Взимку	Влітку
Температура, C^0	22-24	23-25
Відносна вологість, %	40-60	40-60
Швидкість руху повітря, м/с	0,1	0,1 - 0,2

При недотриманні вказаних показників мікроклімату в офісних приміщеннях робочий день для робітників повинен бути скорочений мінімум на 10%.

3.3.5 Електромагнітні випромінювання

Рівень неіонізуючих електромагнітних випромінювань, електростатичних і магнітних полів, а також інтенсивність потоків інфрачервоного та ультрафіолетового випромінювань на робочих місцях повинні відповідати санітарно-гігієнічним вимогам, встановленим у ДСанПіН 3.3.2.007-98 "Державні санітарні норми та правила гігієни праці при роботі з візуальними дисплейними терміналами" та ДСанПіН 3.3.6.096-2002 "Державні санітарні норми та правила щодо фізичних факторів у робочому середовищі ". Важливо зазначити, що дані норми встановлюють гранично допустимі рівні (ГДР) та допустимі рівні (ДР) впливу неіонізуючого випромінювання на людину. Контроль за дотриманням санітарно-гігієнічних вимог до неіонізуючого випромінювання на робочих місцях здійснюється органами державного санітарно-епідеміологічного нагляду.

3.4 Вимоги до робочого місця працівника та його організації

Працівник, який допускається до роботи з комп'ютером, повинен пройти медичний огляд, навчання за спеціальністю, яку він обрав, а також вступний і первинний інструктаж з охорони праці на робочому місці. Далі, кожні пів року, працівники проходять повторні інструктажі з охорони праці та медичні огляди раз на два роки. Робочі місця мають бути організовані так, щоб у поле зору працівника не потрапляли відбиваючі поверхні, вікна та освітлювальні прилади. Робочі місця з моніторами слід розміщувати в глибині приміщення. Не допускається розташування монітору таким чином, щоб працівник сидів обличчям або спиною до вікон, незалежно від типу освітлення, будь то пряме або відбите світло.

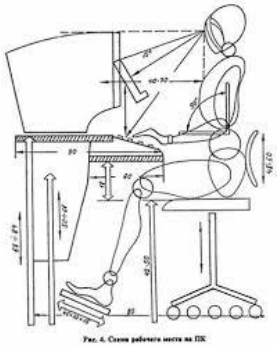


Рисунок 3.1. Схема регулювання робочого столу програміста.

Робочий стіл має регулюватися по висоті в межах 680-800 мм, а ширина столу повинна забезпечувати можливість виконання операцій в зоні досяжності моторного поля.

Розташування екрана ВДТ повинно забезпечувати зручність зорового спостереження у вертикальній площині під кутом +30° до нормальної лінії погляду працівника. Клавіатуру слід розташовувати на поверхні столу на відстані 100-300 мм від краю, зверненого до працівника.

Рекомендовані розміри столу: висота 725 мм, ширина 600-1400 мм, глибина 800-1000 мм. Кожен параметр регулювання має бути легко налаштованим, незалежним і надійно фіксуватися.

ВИСНОВКИ

У процесі виконання дипломної роботи було реалізовано проект розробки 2D-гри у жанрі роуглайк на програмному рушії Unity. Головна мета роботи була закладена у створенні початкової версії гри з процедурно згенерованими рівнями, що забезпечує неповторний та унікальний ігровий досвід при кожному проходженні.

Для виконання цього проекту було досліджено теоретичні основи жанру, його історію, ключові відмінності та особливості, що дозволило визначити основні вимоги до гри та початкових її механік. Було розглянуто ігри, які займають лідируючі позиції цього жанру та взяті основні їх переваги. Також було зроблено аналіз програмного рушія Unity, через що вибрано найбільш ефективні інструменти та методи реалізації роботи.

Процес розробки був розбитий на декілька етапів та покрокових завдань, щоб робота була більш продумана та деталізована, а ігрові механіки були на вищому рівні.

В кінці роботи було проведено тестування гри для виявлення та усунення помилок користування. В результаті виконання проекту було отримано функціональний прототип гри, який є відповідним до стандартів якості індустрії ігор у жанрі роуглайк. Механіки гри, які були реалізовані забезпечують цікавість для гравців різного рівня.

В свою ж чергу використання рушія Unity дало змогу ефективно організувати процес розробки, оптимізувати використані ресурси, а також досягти високої продуктивності гри.

Загалом, виконана робота показала актуальність та перспективність даного жанру, а також показала можливості Unity для створення якісних 2D-ігор. Отриманий результат може бути використаний для майбутнього розвитку проекту, наприклад, додавання нових функцій.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Коноваленко І.В., Програмування мовою C# 6.0, Тернопіль, ТНТУ, 2016 р., 227 ст.
2. Дікінсон К. Оптимізація ігор у Unity 5. Поради і методики, Оптимізація додатків охоплюючи всі аспекти роботи у ігровому движку Unity 3D., Київ, Фабула, 2017 р., 306 ст.
3. Microsoft learn: [Website]. URL: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/covariance-contravariance/ (viewed on: 04.04.2024).
4. Ерік Фрімен, Елізабет Робсон, Берт Бейтс, Кеті Сієрра, Книга Head First. Патерни проектування, Київ, Фабула, 672 ст.
5. Sharpcoderblog: [Веб-сайт]. URL: https://uk.sharpcoderblog.com/blog/introduction-to-interfaces-in-csharp (viewed on: 02.05.2024).
6. Unity: [Website]. URL: https://unity.com/en-us (viewed on: 11.05.2024).
7. C# docs: [Website]. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (viewed on: 21.05.2024).
8. Programm.top: [Website]. URL: https://programm.top/uk/c-sharp/tutorial/ (viewed on: 18.01.2024).
9. Andrew R., Dave M. GAME Architecture and Design – A new Edition, Shelter Island, New York: Manning Publications. 1035 p.
10. Фленов М., Біблія C# 3-е видання., Київ, Фабула, 2016 р. 546 ст.
11. Hocking J. Unity in Action: Multiplatform Game Development in C# with Unity
12. Goldstone W. Unity Game Development Essentials, New York, PACKT, 266 p.
13. Unity: [Website]. URL: https://unity.com/en-us (viewed on: 21.05.2023).

ДОДАТОК А. Лістинг коду основних модулів гри мовою C#

Скрипт Weapon

```
using UnityEngine;
public class Weapon : MonoBehaviour
{
    public float offset;
    private float time;
    public float startTime;

    public GameObject bullet;
    public Transform point;

    void Update()
    {
        Vector3 mousePosition = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        Vector3 difference = mousePosition - transform.position;
        difference.z = 0f;
        float rotateZ = Mathf.Atan2(difference.y, difference.x) * Mathf.Rad2Deg;
        transform.rotation = Quaternion.Euler(0f, 0f, rotateZ + offset);

        // Стрельба
        if (time <= 0f)
        {
            if (Input.GetMouseButtonDown(0))
            {
                Instantiate(bullet, point.position, point.rotation);
                time = startTime;
            }
        }
        else
        {
            time -= Time.deltaTime;
        }
    }
}
```

Скрипт PlayerController

```
using UnityEngine;
using UnityEngine.UI;

public class PlayerController : MonoBehaviour
{

    public float speed;
    Rigidbody2D rb;
    public Text collectedText;
    public static int collectedAmount = 0;
    public GameObject bulletPrefab;
    public float bulletSpeed;
```

```
    private float lastFire;
    public float fireDelay;

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    // Update is called once per frame
    void Update()
    {
        fireDelay = GameController.FireRate;
        speed = GameController.MoveSpeed;
        fireDelay = GameController.FireRate;
        speed = GameController.MoveSpeed;
        float horizontal = Input.GetAxis("Horizontal");
        float vertical = Input.GetAxis("Vertical");

        rb.velocity = new Vector3 (horizontal * speed, vertical * speed, 0);
        collectedText.text = "Item collected: " + collectedAmount;
    }
}
```

Скрипт HelthUIController

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

public class HelthUIController : MonoBehaviour
{
    public GameObject heartContainer;
    private float fillValue;

    void Update()
    {
        fillValue = (float)GameController.Health;
        fillValue = fillValue / GameController.MaxHealth;
        heartContainer.GetComponent<Image>().fillAmount = fillValue;
    }
}
```

Скрипт GameController

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

public class GameController : MonoBehaviour
{
    public static GameController instance;
```

```
private static float health = 6;

private static int maxHealth = 6;

private static float moveSpeed = 5f;

private static float fireRate = 0.5f;

private static float bulletSize = 0.5f;

public static float Health { get => health; set => health = value; }
public static int MaxHealth { get => maxHealth; set => maxHealth = value; }
public static float MoveSpeed { get => moveSpeed; set => moveSpeed = value; }
public static float FireRate { get => fireRate; set => fireRate = value; }
public static float BulletSize { get => bulletSize; set => bulletSize = value; }

public Text healthText;
private void Awake()
{
    if (instance == null)
    {
        instance = this;
    }
}

void Update()
{
    healthText.text = "Health: " + health;
}

public static void DamagePlayer(int damage)
{
    health -= damage;
    if(Health <= 0)
    {
        KillPlayer();
    }
}

public static void HealPlayer(float healAmount)
{
    health = Mathf.Min(maxHealth, health + healAmount);
}
public static void MoveSpeedChange(float speed)
{
    moveSpeed += speed;
}
public static void FireRateChange(float rate)
{
    fireRate -= rate;
```

```
    }
    public static void BulletSizeChange(float size)
    {
        bulletSize += size;
    }

    public static void KillPlayer()
    {

    }
}
```

Скрипт EnemyWeapon

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

public class EnemyWeapon : MonoBehaviour
{
    public float offset;
    private float time;
    public float startTime;

    public GameObject bullet;
    public Transform point;
    public GameObject player; // ссылка на объект игрока

    void Update()
    {
        // Получаем положение игрока
        Vector3 PlayerPosition = player.transform.position;
        // Рассчитываем разницу между положением игрока и позицией объекта (оружия)
        Vector3 difference = PlayerPosition - transform.position;
        // Устанавливаем Z координату разницы на 0
        difference.z = 0f;

        // Рассчитываем угол поворота в градусах
        float rotateZ = Mathf.Atan2(difference.y, difference.x) * Mathf.Rad2Deg;
        // Устанавливаем поворот объекта с учетом смещения
        transform.rotation = Quaternion.Euler(0f, 0f, rotateZ + offset);

        // Стрельба
        if (time <= 0f)
        {
            if (GameObject.FindGameObjectWithTag("Player"))
            {
                Instantiate(bullet, point.position, point.rotation);
                time = startTime;
            }
        }
        else
```

```
        {
            time -= Time.deltaTime;
        }
    }
}
```

Скрипт EnemyState

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

public enum EnemyState
{
    Wander,
    Follow,
    Die,
    Attack
};

public enum EnemyType
{
    Melee,
    Ranged
};

public class EnemyControle : MonoBehaviour
{
    GameObject player;
    public EnemyState currState = EnemyState.Wander;
    public EnemyType enemyType;

    public float range;
    public float speed;
    public float attackRange;
    public float coolDown;

    private bool chooseDir = false;
    private bool dead = false;
    private bool coolDownAttack = false;
    private Vector3 randomDir;
    public GameObject Bullet;

    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player");
    }

    // Update is called once per frame
    void Update()
    {
        switch (currState)
```

```
        {
            case (EnemyState.Wander):
                Wander();
            break;
            case (EnemyState.Follow):
                Follow();
            break;
            case (EnemyState.Die):
            break;
            case (EnemyState.Attack):
                Attack();
            break;
        }
        if(IsPlayerInRange(range) && currState != EnemyState.Die)
        {
            currState = EnemyState.Follow;
        }
        else if(!IsPlayerInRange(range) && currState != EnemyState.Die)
        {
            currState = EnemyState.Wander;
        }
        if (Vector3.Distance(transform.position, player.transform.position) <= attackRange)
        {
            currState = EnemyState.Attack;
        }
    }

    private bool IsPlayerInRange(float range)
    {
        return Vector3.Distance(transform.position, player.transform.position) < range;
    }

    private IEnumerator ChooseDirection()
    {
        chooseDir = true;
        yield return new WaitForSeconds(Random.Range(2f, 8f));
        randomDir = new Vector3(0, 0, Random.Range(0, 360));
        Quaternion nextRotation = Quaternion.Euler(randomDir);
        transform.rotation = Quaternion.Lerp(transform.rotation, nextRotation, Random.Range(0.5f,
2.5f));
        chooseDir = false;
    }
    void Wander()
    {
        if(!chooseDir)
        {
            StartCoroutine(ChooseDirection());
        }
        transform.position += -transform.right * speed * Time.deltaTime;
        if (IsPlayerInRange(range))
```

```
        {
            currState = EnemyState.Follow;
        }
    }
    void Follow()
    {
        transform.position = Vector2.MoveTowards(transform.position, player.transform.position,
speed * Time.deltaTime);
    }
    void Attack()
    {
        if (!coolDownAttack)
        {
            switch (enemyType)
            {
                case (EnemyType.Melee):
                    GameController.DamagePlayer(1);
                    StartCoroutine(CoolDown());
                    break;
                case (EnemyType.Ranged):

                  // GameObject bullet = Instantiate(Bullet, transform.position, Quaternion.identity) as
GameObject;
                    //bullet.GetComponent<Bullet>().GetPlayer(player.transform);
                   // bullet.AddComponent<Rigidbody2D>().gravityScale = 0;
                    //bullet.GetComponent<Bullet>().isEnemyBullet = true;
                    //StartCoroutine(CoolDown());
                    break;
            }
        }
    }
    private IEnumerator CoolDown()
    {
        coolDownAttack = true;
        yield return new WaitForSeconds(coolDown);
        coolDownAttack = false;
    }

    public void Death()
    {
        Destroy(gameObject);
    }
}
```

Скрипт Enemy

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

public class Enemy : MonoBehaviour
{
```

```
    public int health;

    void Update()
    {
        if (health <= 0)
        {
            Destroy(gameObject);
        }
    }
    public void TakeDamage(int damage)
    {
        health -= damage;
    }
}
```

Скрипт CollectionController

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

[System.Serializable]
public class Item
{
    public string name;
    public string description;
    public Sprite itemImage;
}


public class CollectionController : MonoBehaviour
{
    public Item item;
    public float healthChange;
    public float moveSpeedChange;
    public float attackSpeedChange;
    public float bulletSizeChange;

    void Start()
    {
        GetComponent<SpriteRenderer>().sprite = item.itemImage;
        Destroy(GetComponent<PolygonCollider2D>());
        gameObject.AddComponent<PolygonCollider2D>();
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Player")
        {
            PlayerController.collectedAmount++;
            GameController.HealPlayer(healthChange);
            GameController.MoveSpeedChange(moveSpeedChange);
```

```
            GameController.FireRateChange(attackSpeedChange);
            GameController.BulletSizeChange(bulletSizeChange);
            Destroy(gameObject);
        }
    }

}
```

Скрипт CameraController

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

public class CameraController : MonoBehaviour
{
    public static CameraController instance;
    public Room currRoom;
    public float moveSpeedWhenRoomChange;

    void Awake()
    {
        instance = this;
    }

    // Update is called once per frame
    void Update()
    {
        UpdatePosition();
    }
    void UpdatePosition()
    {
        if (currRoom == null)
        {
            return;
        }

        Vector3 targetPos = GetCameraTargetPosition();

        transform.position = Vector3.MoveTowards(transform.position, targetPos, Time.deltaTime *
moveSpeedWhenRoomChange);
    }

    Vector3 GetCameraTargetPosition()
    {
        if (currRoom == null)
        {
            return Vector3.zero;
        }

        Vector3 targetPos = currRoom.GetRoomCentre();
        targetPos.z = transform.position.z;
```

```
        return targetPos;
    }
    public bool IsSwitchingScene()
    {
        return transform.position.Equals(GetCameraTargetPosition()) == false;
    }
}
```

Скрипт Bullet

```
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
using static UnityEditor.ShaderGraph.Internal.KeywordDependentCollection;

public class Bullet : MonoBehaviour
{
    public float speed;
    public float distance;
    public int damage;
    //public GameObject destroyEffect;
    //public GameObject bloodSplash;
    public LayerMask layerMask;


    //public float lifeTime;
    public bool isEnemyBullet = false;

    private Vector2 lastPos;
    private Vector2 curPos;
    private Vector2 playerPos;
    // Start is called before the first frame update
    void Start()
    {
        //StartCoroutine(DeathDelay());
        if (!isEnemyBullet)
        {
            transform.localScale = new Vector2(GameController.BulletSize,
GameController.BulletSize);
        }
    }

    void Update()
    {
        RaycastHit2D other = Physics2D.Raycast(transform.position, transform.up, distance,
layerMask);
        if (other.collider != null)
        {
            if (other.collider.CompareTag("Enemy"))
            {
```

```
            other.collider.GetComponent<Enemy>().TakeDamage(damage);
            Destroy();
        }

        if (other.collider.CompareTag("Ground"))
        {
            Destroy();
        }
    }
    transform.Translate(Vector2.up * speed * Time.deltaTime);

    if (isEnemyBullet)
    {
        curPos = transform.position;
        transform.position = Vector2.MoveTowards(transform.position, playerPos, 5f *
Time.deltaTime);
        if (curPos == lastPos)
        {
            Destroy(gameObject);
        }
        lastPos = curPos;
    }
  }
  private void Destroy()
  {
    // Instantiate(destroyEffect, transform.position, Quaternion.identity);
    //Instantiate(bloodSplash, transform.position, Quaternion.identity);
    Destroy(gameObject);
  }
  public void GetPlayer(Transform player)
  {
    playerPos = player.position;
  }

  void OnTriggerEnter2D(Collider2D col)
  {
    /*if (col.tag == "Enemy" && !isEnemyBullet)
    {
        col.gameObject.GetComponent<EnemyControle>().Death();
        Destroy(gameObject);
    }*/

    if (col.tag == "Player" && isEnemyBullet)
    {
        GameController.DamagePlayer(1);
        Destroy(gameObject);
    }
  }

  /*IEnumerator DeathDelay()
```

```
    {
        yield return new WaitForSeconds(lifeTime);
        Destroy(gameObject);
    }*/
}
```

Скрипт RoomController

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using System.Linq;
public class RoomInfo
{
    public string name;
    public int X;
    public int Y;
}
public class RoomController : MonoBehaviour
{
    public static RoomController instance;

    string currentWorldName = "Basement";

    RoomInfo currentLoadRoomData;

    Room currRoom;

    Queue<RoomInfo> loadRoomQueue = new Queue<RoomInfo>();

    public List<Room> loadedRooms = new List<Room>();

    bool isLoadingRoom = false;
    bool spawnedBossRoom = false;
    bool updatedRooms = false;

    void Awake()
    {
        instance = this;
    }

    void Update()
    {
        UpdateRoomQueue();
    }

    void UpdateRoomQueue()
    {
        if (isLoadingRoom)
        {
            return;
```

```
        }

        if (loadRoomQueue.Count == 0)
        {
            if (!spawnedBossRoom)
            {
                StartCoroutine(SpawnBossRoom());
            }
            else if (spawnedBossRoom && !updatedRooms)
            {
                foreach (Room room in loadedRooms)
                {
                    room.RemoveUnconnectedDoors();
                }
                //UpdateRooms();
                updatedRooms = true;
            }
            return;
        }

        currentLoadRoomData = loadRoomQueue.Dequeue();
        isLoadingRoom = true;

        StartCoroutine(LoadRoomRoutine(currentLoadRoomData));
    }


    IEnumerator SpawnBossRoom()
    {
        spawnedBossRoom = true;
        yield return new WaitForSeconds(0.5f);
        if (loadRoomQueue.Count == 0)
        {
            Room bossRoom = loadedRooms[loadedRooms.Count - 1];
            Room tempRoom = new Room(bossRoom.X, bossRoom.Y);
            Destroy(bossRoom.gameObject);
            var roomToRemove = loadedRooms.Single(r => r.X == tempRoom.X && r.Y ==
tempRoom.Y);
            loadedRooms.Remove(roomToRemove);
            LoadRoom("End", tempRoom.X, tempRoom.Y);
        }
    }


    public void LoadRoom(string name, int x, int y)
    {
        if (DoesRoomExist(x, y) == true)
        {
            return;
        }
```

```
        RoomInfo newRoomData = new RoomInfo();
        newRoomData.name = name;
        newRoomData.X = x;
        newRoomData.Y = y;

        loadRoomQueue.Enqueue(newRoomData);
    }

    IEnumerator LoadRoomRoutine(RoomInfo info)
    {
        string roomName = currentWorldName + info.name;

        AsyncOperation loadRoom = SceneManager.LoadSceneAsync(roomName,
LoadSceneMode.Additive);

        while (loadRoom.isDone == false)
        {
            yield return null;
        }
    }
    public void RegisterRoom(Room room)
    {
        if (!DoesRoomExist(currentLoadRoomData.X, currentLoadRoomData.Y))
        {
            room.transform.position = new Vector3(
                currentLoadRoomData.X * room.Width,
                currentLoadRoomData.Y * room.Height,
                0
            );

            room.X = currentLoadRoomData.X;
            room.Y = currentLoadRoomData.Y;
            room.name = currentWorldName + "-" + currentLoadRoomData.name + " " + room.X + ",
" + room.Y;
            room.transform.parent = transform;

            isLoadingRoom = false;

            if (loadedRooms.Count == 0)
            {
                CameraController.instance.currRoom = room;
            }

            loadedRooms.Add(room);
        }
        else
        {
            Destroy(room.gameObject);
            isLoadingRoom = false;
```

```
        }

    }

    public bool DoesRoomExist(int x, int y)
    {
        return loadedRooms.Find(item => item.X == x && item.Y == y) != null;
    }

    public Room FindRoom(int x, int y)
    {
        return loadedRooms.Find(item => item.X == x && item.Y == y);
    }
    public string GetRandomRoomName()
    {
        string[] possibleRooms = new string[] {
            "Empty",
            "Basic1"
        };

        return possibleRooms[Random.Range(0, possibleRooms.Length)];
    }
    public void OnPlayerEnterRoom(Room room)
    {
        CameraController.instance.currRoom = room;
        currRoom = room;

    }

}
```

Скрипт Room

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

public class Room : MonoBehaviour
{

    public int Width;
    public int Height;
    public int X;
    public int Y;

    private bool updatedDoors = false;

    public Room(int x, int y)
    {
        X = x;
        Y = y;
    }
```

```
public Door leftDoor;
public Door rightDoor;
public Door topDoor;
public Door bottomDoor;

public List<Door> doors = new List<Door>();

// Start is called before the first frame update
void Start()
{
    if (RoomController.instance == null)
    {
        Debug.Log("You pressed play in the wrong scene!");
        return;
    }

    Door[] ds = GetComponentsInChildren<Door>();
    foreach (Door d in ds)
    {
        doors.Add(d);
        switch (d.doorType)
        {
            case Door.DoorType.right:
                rightDoor = d;
                break;
            case Door.DoorType.left:
                leftDoor = d;
                break;
            case Door.DoorType.top:
                topDoor = d;
                break;
            case Door.DoorType.bottom:
                bottomDoor = d;
                break;
        }
    }

    RoomController.instance.RegisterRoom(this);
}

void Update()
{
    if (name.Contains("End") && !updatedDoors)
    {
        RemoveUnconnectedDoors();
        updatedDoors = true;
    }
}
```

```
public void RemoveUnconnectedDoors()
{
    Debug.Log("removing doors");
    foreach (Door door in doors)
    {
        switch (door.doorType)
        {
            case Door.DoorType.right:
                if (GetRight() == null)
                    door.gameObject.SetActive(false);
                break;
            case Door.DoorType.left:
                if (GetLeft() == null)
                    door.gameObject.SetActive(false);
                break;
            case Door.DoorType.top:
                if (GetTop() == null)
                    door.gameObject.SetActive(false);
                break;
            case Door.DoorType.bottom:
                if (GetBottom() == null)
                    door.gameObject.SetActive(false);
                break;
        }
    }
}

public Room GetRight()
{
    if (RoomController.instance.DoesRoomExist(X + 1, Y))
    {
        return RoomController.instance.FindRoom(X + 1, Y);
    }
    return null;
}
public Room GetLeft()
{
    if (RoomController.instance.DoesRoomExist(X - 1, Y))
    {
        return RoomController.instance.FindRoom(X - 1, Y);
    }
    return null;
}
public Room GetTop()
{
    if (RoomController.instance.DoesRoomExist(X, Y + 1))
    {
        return RoomController.instance.FindRoom(X, Y + 1);
    }
    return null;
```

```
    }
    public Room GetBottom()
    {
        if (RoomController.instance.DoesRoomExist(X, Y - 1))
        {
            return RoomController.instance.FindRoom(X, Y - 1);
        }
        return null;
    }


    void OnDrawGizmos()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireCube(transform.position, new Vector3(Width, Height, 0));
    }

    public Vector3 GetRoomCentre()
    {
        return new Vector3(X * Width, Y * Height);
    }

    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.tag == "Player")
        {
            RoomController.instance.OnPlayerEnterRoom(this);
        }
    }
}
```

Скрипт DungeonGenerator

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

public class DungeonGenerator : MonoBehaviour
{
    public DungeonGenerationData dungeonGenerationData;
    private List<Vector2Int> dungeonRooms;

    private void Start()
    {
        dungeonRooms = DungeonCrawlerController.GenerateDungeon(dungeonGenerationData);
        SpawnRooms(dungeonRooms);
    }

    private void SpawnRooms(IEnumerable<Vector2Int> rooms)
    {
        RoomController.instance.LoadRoom("Start", 0, 0);
        foreach (Vector2Int roomLocation in rooms)
```

```
        {
            RoomController.instance.LoadRoom("Empty", roomLocation.x, roomLocation.y);

        }
    }
}
```

Скрипт Direction

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

[CreateAssetMenu(fileName = "DungeonGenerationData.asset", menuName =
"DungeonGenerationData/Dungeon Data")]
public class DungeonGenerationData : ScriptableObject
{
    public int numberOfCrawlers;
    public int iterationMin;
    public int iterationMax;
}
```

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

public enum Direction
{
    up = 0,
    left = 1,
    down = 2,
    right = 3
};
```

Скрипт DungeonCrawlerController

```
public class DungeonCrawlerController : MonoBehaviour
{

    public static List<Vector2Int> positionsVisited = new List<Vector2Int>();
    private static readonly Dictionary<Direction, Vector2Int> directionMovementMap = new
Dictionary<Direction, Vector2Int>
    {
        {Direction.up, Vector2Int.up},
        {Direction.left, Vector2Int.left},
        {Direction.down, Vector2Int.down},
        {Direction.right, Vector2Int.right}
    };

    public static List<Vector2Int> GenerateDungeon(DungeonGenerationData dungeonData)
    {
        List<DungeonCrawler> dungeonCrawlers = new List<DungeonCrawler>();

        for (int i = 0; i < dungeonData.numberOfCrawlers; i++)
```

```
        {
            dungeonCrawlers.Add(new DungeonCrawler(Vector2Int.zero));
        }

        int iterations = Random.Range(dungeonData.iterationMin, dungeonData.iterationMax);

        for (int i = 0; i < iterations; i++)
        {
            foreach (DungeonCrawler dungeonCrawler in dungeonCrawlers)
            {
                Vector2Int newPos = dungeonCrawler.Move(directionMovementMap);
                positionsVisited.Add(newPos);
            }
        }

        return positionsVisited;
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

Скрипт DungeonCrawler

```
public class DungeonCrawler : MonoBehaviour
{
    public Vector2Int Position { get; set; }
    public DungeonCrawler(Vector2Int startPos)
    {
        Position = startPos;
    }

    public Vector2Int Move(Dictionary<Direction, Vector2Int> directionMovementMap)
    {
        Direction toMove = (Direction)Random.Range(0, directionMovementMap.Count);
        Position += directionMovementMap[toMove];
        return Position;
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Door : MonoBehaviour
{
    public enum DoorType
    {
        left, right, top, bottom
    }

    public DoorType doorType;
}
```

Скрипт CameraController

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

public class CameraController : MonoBehaviour
{
    public static CameraController instance;
    public Room currRoom;
    public float moveSpeedWhenRoomChange;

    void Awake()
    {
        instance = this;
    }

    // Update is called once per frame
    void Update()
    {
        UpdatePosition();
    }
    void UpdatePosition()
    {
        if (currRoom == null)
        {
            return;
        }

        Vector3 targetPos = GetCameraTargetPosition();

        transform.position = Vector3.MoveTowards(transform.position, targetPos, Time.deltaTime *
moveSpeedWhenRoomChange);
    }

    Vector3 GetCameraTargetPosition()
    {
        if (currRoom == null)
        {
            return Vector3.zero;
        }

        Vector3 targetPos = currRoom.GetRoomCentre();
        targetPos.z = transform.position.z;

        return targetPos;
    }
    public bool IsSwitchingScene()
    {
        return transform.position.Equals(GetCameraTargetPosition()) == false;
    }
}
```

Скрипт Bullet

```
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
using static UnityEditor.ShaderGraph.Internal.KeywordDependentCollection;

public class Bullet : MonoBehaviour
{
    public float speed;
    public float distance;
    public int damage;
    //public GameObject destroyEffect;
    //public GameObject bloodSplash;
    public LayerMask layerMask;


    //public float lifeTime;
    public bool isEnemyBullet = false;

    private Vector2 lastPos;
    private Vector2 curPos;
    private Vector2 playerPos;
    // Start is called before the first frame update
    void Start()
    {
        //StartCoroutine(DeathDelay());
        if (!isEnemyBullet)
        {
            transform.localScale = new Vector2(GameController.BulletSize,
GameController.BulletSize);
        }
    }

    void Update()
    {
        RaycastHit2D other = Physics2D.Raycast(transform.position, transform.up, distance,
layerMask);
        if (other.collider != null)
        {
            if (other.collider.CompareTag("Enemy"))
            {
                other.collider.GetComponent<Enemy>().TakeDamage(damage);
                Destroy();
            }

            if (other.collider.CompareTag("Ground"))
            {
                Destroy();
            }
```

```
        }
        transform.Translate(Vector2.up * speed * Time.deltaTime);

        if (isEnemyBullet)
        {
            curPos = transform.position;
            transform.position = Vector2.MoveTowards(transform.position, playerPos, 5f *
Time.deltaTime);
            if (curPos == lastPos)
            {
                Destroy(gameObject);
            }
            lastPos = curPos;
        }
    }
    private void Destroy()
    {
        // Instantiate(destroyEffect, transform.position, Quaternion.identity);
        //Instantiate(bloodSplash, transform.position, Quaternion.identity);
        Destroy(gameObject);
    }
    public void GetPlayer(Transform player)
    {
        playerPos = player.position;
    }

    void OnTriggerEnter2D(Collider2D col)
    {
        /*if (col.tag == "Enemy" && !isEnemyBullet)
        {
            col.gameObject.GetComponent<EnemyControle>().Death();
            Destroy(gameObject);
        }*/

        if (col.tag == "Player" && isEnemyBullet)
        {
            GameController.DamagePlayer(1);
            Destroy(gameObject);
        }
    }

    /*IEnumerator DeathDelay()
    {
        yield return new WaitForSeconds(lifeTime);
        Destroy(gameObject);
    }*/
}
```

ДОДАТОК Б. Слайди мультимедійної презентації

Розробка комп'ютерної 2D-гри в жанрі роуглайк за допомогою рушія Unity

ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТА 4РП-07: НЕЙКО ІЛЛІ ІГОРОВИЧА

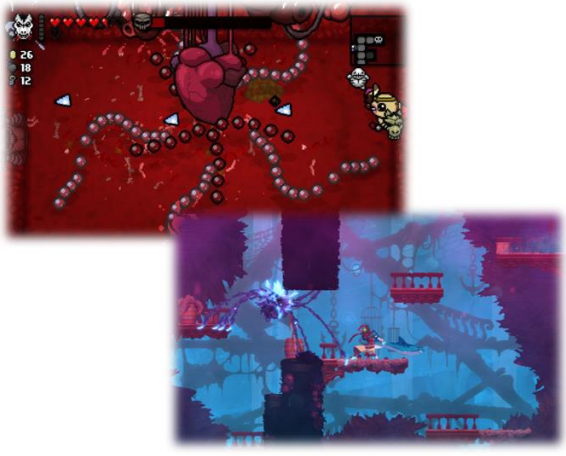
КЕРІВНИК: ДЖАБРАІЛОВ Д.В.

Особливості ігрового жанру 2D роуглайк

Жанр 2D роуглайк (roguelike) займає унікальне місце у світі відеоігор завдяки своїм характерним особливостям та механікам, що створюють унікальний ігровий досвід.

До особливостей ігрового жанру відносять:

- Випадкова генерація рівнів
- Постійна смерть (permadeath)
- Покроковий геймплей
- Висока складність та покарання за помилки
- Різноманіття класів та предметів
- Прогресія через невдачі
- Атмосферний дизайн та сюжетні елементи

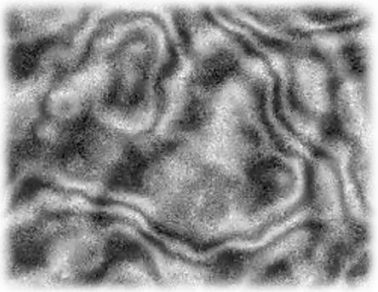


Принципи процедурної генерації в ігрових проектах

Процедурна генерація є методикою створення контенту в реальному часі за допомогою алгоритмів, замість ручної розробки кожного елемента гри.

Принципи процедурної генерації в ігрових проектах:

- Шум Перліна
- Алгоритмічний підхід
- Випадковість з контролем
- Адаптивність до дій гравця
- Серединне ділення



Шум Перліна

Особливості програмного рушія Unity та причини його вибору для розробки проекту

Програмний рушій Unity є одним з найпопулярніших інструментів для розробки відеоігор завдяки своїм потужним можливостям, зручності використання та широкому набору інструментів.

Особливості Unity:

- Зручність використання та навчання
- Підтримка багатоплатформеності
- Потужний редактор сцен
- Розширюваність та інтеграції
- Скриптинг на C#

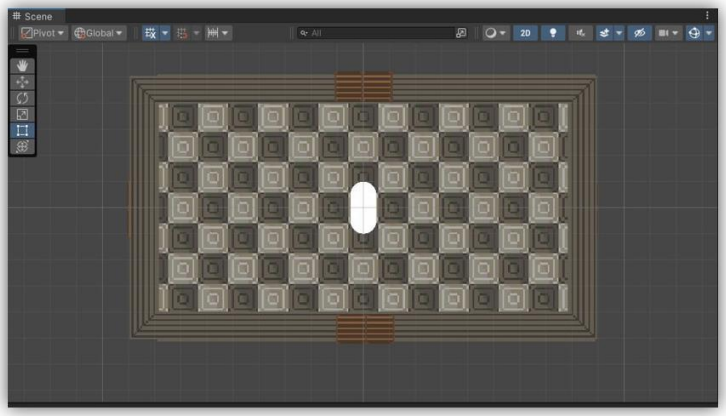
Причини вибору Unity:

- Універсальність та гнучкість
- Ефективність розробки
- Активна спільнота та підтримка
- Економічна вигода
- Широкі можливості

Unity

Особливості ігрових механік розроблюємого проекту

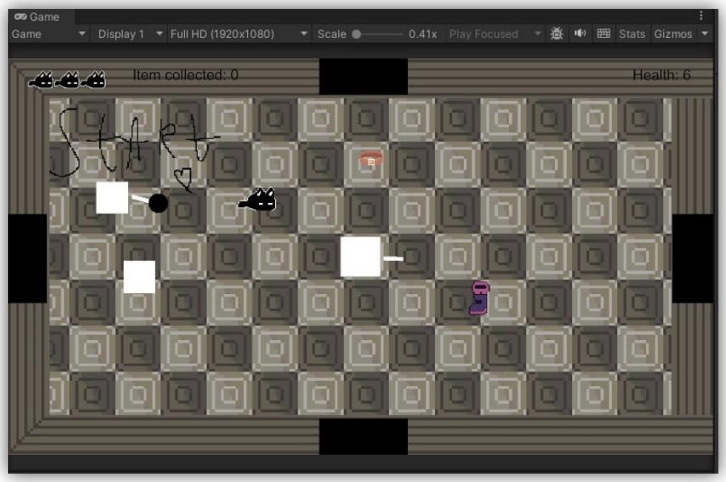
Після аналізу схожих проектів, було вирішено додати деякі особливості в свій проект. Я вирішив додати в генерацію рівня одну особливу кімнату - кімнату з босом, яка генерується в кінці рівня. Ще з особливостей є тим, що гравець стріляє туди, куди направлений курсор, що забезпечує ліпший контроль над стрільбою.



Огляд основних елементів ігрового процесу

З основних елементів в проекті присутнє наступне:

- Випадкова поява предметів та противників на рівні
- Предмети, які впливають на характеристики гравця
- Генерація рівня



Принцип роботи основних елементів ігрового процесу

- Генерація предметів та ворогів реалізована так, що кожна кімната розбивається на сітку, та випадково обираються квадрати, де будуть генеруватися предмети або ворог.

- Реалізація предметів: коли гравець збігається з колайдером предмета, то спочатку визначається тип предмету, потім він змінює значення параметрів гравця та видаляється зі сцени.

- Генерація рівня. Спочатку генеруються випадкові координати для кімнат, потім створюються кімнати на цих координатах. Після цього проходить перевірка кімнат на з’єднання з іншими., та в місцях з’єднання залишаються двері.

Етапи реалізації основних елементів ігрового процесу

- Основні етапи реалізації включали в себе:

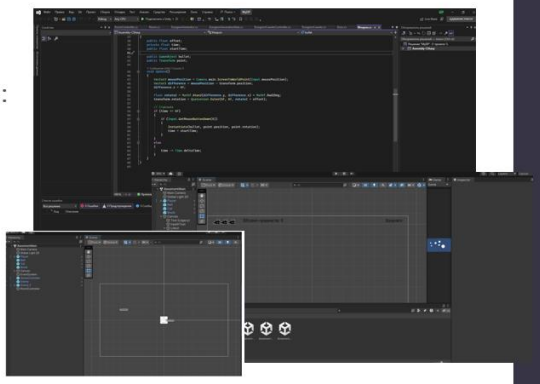
-Формування концептів та думок

-Створення проекту в Unity

-Написання скриптів

-Впровадження скриптів в проект

-Тестування та виправлення помилок



Хід тестування гри

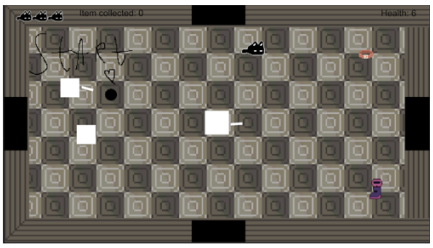
- На етапі тестування були знайдені

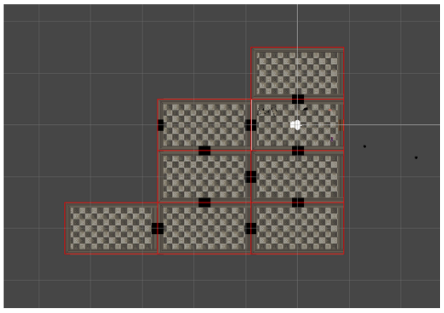
деякі баги, які були виправленні одразу:

-Виправлено баг з неправильним розташуванням кімнати з босом.

-Виправлено баг з неможливістю вбити гравця ворогом.

Скріни готової гри





ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Нейко Іллі Ігоровича

(прізвище, ім`я та по батькові)

Спеціальність: 121 "Інженерія програмного забезпечення"

Освітня програма: «Розробка програмного забезпечення»

Тема дипломного проекту: Розробка 2D-гри у жанрі роуглайк на програмному рушії Unity

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ

а) обсяг і якість виконання проекту (графічного матеріалу і розрахунково-пояснювальної записки) Дипломний проект виконано відповідно технічному завданню. Пояснювальна записка містить 87 сторінок. У пояснювальній записці виконано опис предметної області, способи реалізації гри в жанрі роуглайк. Проведено проектування та реалізацію всіх необхідних систем гри. Графічна частина складається з 10 слайдів мультимедійної презентації, які передбачені технічним завданням. Якість виконання пояснювальної записки та графічної частини добра, розробку виконано в повному обсязі.

б) самостійність роботи над проектом: Протягом всього строку дипломного проектування та переддипломної практики здобувач освіти Нейко І.І. поступово та послідовно виконував всі етапи розробки. Всі роботи здобувач освіти виконував самостійно, з оглядом на рекомендації керівника.

в) теоретична підготовка випускника (випускниці): Здобувач освіти Нейко І.І. під час роботи над дипломним проектом вивчив достатню кількість літературних джерел та матеріалів за даною тематикою.

Вважаю, що теоретична підготовка дипломника добра і він готовий до захисту дипломного проекту

г) вміння розв'язувати виробничі та конструкторські питання _____

Під час дипломного проектування здобувач освіти Нейко І.І. мав змогу самостійно приймати рішення з реалізації елементів розробляємої гри, та показав вміння організовано працювати над поставленим завданням, складати схеми та проводити розробку коду за допомогою актуальних для теми комп'ютерних програмних засобів.

Оцінка розрахункової частини Відмінно

Оцінка графічної частини Добре

Загальна оцінка Відмінно

Прізвище, ім'я, по батькові керівника дипломного проекту

Джабраілов Дмитро Володимирович

Місце роботи і посада керівника дипломного проекту

ВСП "Одеський технічний фаховий коледж ОНТУ", викладач комісії комп'ютерних технологій та програмної інженерії

Підпис _____

«10» 06 2024 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект (роботу) здобувача (здобувачки) освіти

відділення комп'ютерних систем

Нейко Іллі Ігоровича

(прізвище, ім`я та по батькові)

Спеціальність 121 " Інженерія програмного забезпечення "

Освітня програма «Розробка програмного забезпечення»

Керівник дипломного проекту (роботи) Джабраілов Дмитро Володимирович

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) *Розробка 2D-гри у жанрі роуглайк на програмному рушії Unity*

Обсяг розрахунково-пояснювальної записки 79 сторінок

Обсяг графічної (презентаційної) частини 12 аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту (роботи) завданню

Представлений на рецензію дипломний проект повністю відповідає меті проектування та технічному завданню. Тематика дипломного проекту є актуальною для своєї галузі та присвячена питанням створення ігрових продуктів в цілому та розробці ігор на ігровому програмному рушії Unity, в цілому.

б) характеристика виконання кожного розділу дипломного проекту (роботи)

Дипломний проект складається зі вступу, трьох розділів, висновків, переліку використаних джерел. У технологічному розділі розглянуті питання проблематики розробки гри на ігровому програмному рушії Unity, сформовано вимоги до гри згідно до теми дипломного проекту та завданню, виконано проектування основних аспектів розробляємої гри. За допомогою відповідного програмного забезпечення реалізовані всі намічені зміни до ігрового процесу

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту (роботи) Графічна частина виконана на достатньо високому рівні у вигляді презентації із використанням офісного пакету Microsoft PowerPoint та Visio. Пояснювальна записка виконана акуратно та у відповідності до норм оформлення документів із використанням офісного пакету Microsoft Word. Загальна якість виконання документації – добра, академічного плагіату у роботі не виявлено

г) перелік позитивних якостей дипломного проекту (роботи)

1. Описано процес виконання розробки гри;
2. Виконано проектування елементів гри із поясненнями на схемах та за допомогою коду;
3. Розроблено функціонуючу гру за допомогою відповідних інструментів розробки.

д) основні недоліки дипломного проекту (роботи)

1. Гра має неоднорідне графічне виконання, графічну частину проекту недостатньо пропрацьовано;
2. З пояснювальної записки не зовсім зрозуміло, як працює процедурна генерація рівнів;
3. У роботі присутні деякі помилки оформлення.

Оцінка розрахункової частини Добре

Оцінка графічної частини Добре

Загальна оцінка Добре

Прізвище, ім'я, по батькові рецензента к.т.н. Селіванова Алла Віталіївна

Місце роботи і посада рецензента Одеський національний технологічний університет, декан факультету комп'ютерної інженерії, програмування та кіберзахисту

Підпис:

«18» 06 2024 р.

UNICHECK
by Turnitin

Ім'я користувача:
Катерина Григоріївна Краснокутська

ID перевірки:
1016338873

Дата перевірки:
09.06.2024 18:35:31 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
09.06.2024 20:09:02 EEST

ID користувача:
100011688

Назва документа: **4РП-07_Нейко**

Кількість сторінок: **43** Кількість слів: **6613** Кількість символів: **47402** Розмір файлу: **1.85 MB** ID файлу: **1016139946**

11.3%
Схожість

Найбільша схожість: 4.6% з Інтернет-джерелом (https://ela.kpi.ua/handle/123456789/49671)

11.3% Джерела з Інтернету	211	..... Сторінка 45

Не знайдено джерел з Бібліотеки

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

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

Замінені символи	7

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНГО ДИПЛОМНОГО ПРОЕКТА
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Нейко Ілля Ігорович,
здобувач освіти гр. 4РП-07, та

Джабраілов Дмитро Володимирович,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до випускного дипломного проекту фахового молодшого бакалавра на тему:

«Розробка 2D-гри у жанрі роуглайк на програмному рушії Unity» (автор роботи – Нейко І.І., керівник роботи – Джабраілов Д.В.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2024 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи, і даємо згоду на обробку персональних даних.

Виконавець _____ / Нейко І.І./

Керівник _____ / Джабраілов Д.В./

«10» 06 2024 р.