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.07.10.000.ДП

ЛАНОВЕНКО ЮРІЯ ОЛЕКСАНДРОВИЧА

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 слешер на програмному рушії Unity

Проектний матеріал складається з пояснювальної записки на 77 сторінках та графічного (презентаційного) матеріалу на 10 аркушах (слайдах)

Дипломник _____ (Лановенко Ю.О.)

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 «17» 06 2024 р. Протокол ЕК № 1

Оцінка ЕК 5(відмінно)/90 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 *Лановенко Юрію Олександровичу*

(прізвище, ім'я, по батькові)

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

затверджена наказом по коледжу від "02" 11 202 3 р. № 244-А2-ОД

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; Реалізація основних ігрових механік комп'ютерної 2D-гри в жанрі слешер; Гра повинна мати основні признаки жанру 2D слешер; Гра повинна мати інтерфейс; Гравець повинен мати змогу пересуватись по ігровому простору та взаємодіяти з неігровими персонажами.*

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

Аналіз ігрового жанру 2D слешер; Аналітичний огляд програмних рушіїв з умовно безкоштовним доступом; Формування концепту ігрового процесу 2D-гри в жанрі слешер; 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 слешер; 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	*Іванченков 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 *Лановенко Ю.О.* _____
(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 слешер*	*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. 10 000. 00 ДП	Дипломний проект		
А4			РП 07. 10 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	РП 07. 10 000. 00 ДП		
Розробив		Лановенко Ю.О.		10.06	Розробка 2D-гри у жанрі слешер на програмному рушії Unity	Літ.	Аркуш 4 / Аркушів 77
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

Ігрова індустрія займає значне місце в сучасному світі інформаційних технологій. Вона об'єднує розробників, художників, дизайнерів та гравців, створюючи величезні віртуальні всесвіти та незабутні враження. Від аркадних автоматів до високотехнологічних комп'ютерних та консольних ігор, ця галузь постійно розвивається і вдосконалюється. Індустрія ігор є фінансово перспективною галуззю, приносячи багатомільйонні прибутки, щороку, та продовжуючи зростати і розвиватись.

Програмний продукт, що розроблятиметься протягом дипломного проекту представляє собою 2D гру в жанрі слешер, яка поєднує в собі основні ознаки типового проекту в цьому жанрі. Слешери є одними із найбільш динамічних та захоплюючих жанрів ігровій індустрії, а актуальність розробки таких проектів обумовлена високим попитом на екшен-гри, які дозволяють гравцям відчути себе частиною захоплюючих битв і пригод. Такі проекти не тільки розважають, але й сприяють розвитку реакції, стратегічного мислення та здатності приймати швидкі рішення.

Для розробки даного проекту було обрано ігровий рушій Unity. Unity є одним з найбільш популярних і потужних інструментів для створення 2D та 3D ігор. Він пропонує широкий спектр можливостей, таких як підтримка різних платформ, багатий набір інструментів для роботи з анімацією, фізикою та графікою, а також доступ до великої спільноти розробників. Крім цього, Unity відомий своєю доступністю для початківців та високою ефективністю для професійних проектів, що робить його ідеальним вибором для реалізації цього дипломного проекту.

Головною метою цього дипломного проекту є отримання і закріплення навичок роботи над проектом, від розробки концепту і планування робот, до реалізації механік ігрового процесу.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Аналіз ігрового жанру «2D слешер»

Ігрова індустрія сьогодні є однією з найдинамічніших та найприбутковіших галузей розваг у світі. Вона охоплює широкий спектр жанрів і стилів, що забезпечують різноманітні ігрові враження для всіх типів гравців. Від простих казуальних ігор до складних ігрових світів з глибоким сюжетом і високоякісною графікою. Відеоігри стали невід'ємною частиною сучасної культури розваг.

Жанри відеоігор надзвичайно різноманітні. Серед них можна виділити екшн-ігри, які характеризуються швидким темпом та інтенсивним ігровим процесом, як у популярних франшизах Call of Duty, Assassin's Creed та Doom. Пригодницькі ігри зосереджені на сюжеті, дослідженні світу і взаємодії з персонажами, з прикладами таких серій як The Legend of Zelda, Uncharted та Tomb Raider. Рольові ігри (RPG) дозволяють гравцям зануритися у вигадані світи, створювати і розвивати персонажів, виконувати квести, як у The Elder Scrolls, Final Fantasy та Mass Effect.

Стратегії вимагають від гравців планування та управління ресурсами, вони можуть бути покроковими або реального часу, прикладами є серії StarCraft, Civilization та Total War. Симулятори надають можливість керувати різними аспектами реального життя або вигаданих ситуацій, вони можуть бути дуже реалістичними або жартівливими, як The Sims, SimCity та Microsoft Flight Simulator. Спортивні ігри відтворюють різні види спорту, від футболу та баскетболу до гонок і гольфу, популярні франшизи включають FIFA, NBA 2K та Gran Turismo. Кожен з цих жанрів має свої унікальні особливості та механіки, які задовольняють різні вподобання гравців, забезпечуючи їм захоплюючі та різноманітні ігрові досвіди.

Також ігрова індустрія тісно пов'язана з іншими галузями розваг, такими як кіно, телебачення та музика. Часто відеоігри створюються на основі популярних фільмів та серіалів, а також навпаки – фільми та серіали на основі відеоігор. Музичні композиції, створені для відеоігор, іноді стають популярними самі по

собі, а індустрія кіберспорту (електронного спорту) приваблює мільйони глядачів по всьому світу, подібно до традиційних спортивних подій.

Слешер (від англ. slash), також відомий як Hack’n’Slash (хек-н-слеш), є жанром ігор, де гравець керує персонажем, який протистоїть великій кількості ворогів, зазвичай використовуючи зброю ближнього бою, зокрема холодну зброю. Цей жанр відрізняється швидким темпом гри, видовищними боями та загальним фокусом на бойовій системі.

Жанр слешерів виник ще за часів аркадних автоматів з такими класичними іграми, як "Double Dragon" (1987) та "Final Fight" (1989), які запропонували гравцям можливість битися з натовпом ворогів, використовуючи прості комбінації атак. Ці ігри стали першими кроками до створення більш інтенсивних бойових сцен. Далі, на рисунку 1.1, приведений скриншот гри «Final Fight»:

Рисунок 1.1. Кадр гри «Final Fight»

З роками слешери еволюціонували, виходячи за рамки простого екшену і пропонуючи цікаві, насичені багатогранні ігрові світи. Сьогодні багато успішних проектів у жанрі хек-н-слеш користуються популярністю в усьому світі та цінуються геймерами за їх культурний та технологічний внесок у ігрову індустрію. Прикладами таких проектів є серії ігор, як Devil May Cry, God of War,

Bayonetta та інші.

Сучасні слешери часто включають елементи рольових ігор (RPG), дозволяючи гравцям розвивати персонажа, покращувати його здібності та налаштовувати зброю і спорядження. Це додає глибини ігровому процесу, роблячи його більш захоплюючим та індивідуальним для кожного гравця. Наприклад, у грі "God of War" (2018) гравці можуть не лише битися з ворогами, але й досліджувати великий відкритий світ, взаємодіяти з персонажами та виконувати різноманітні завдання.

Технологічний прогрес також суттєво вплинув на розвиток жанру слешерів. Завдяки новим графічним технологіям, анімації та штучному інтелекту, сучасні ігри можуть запропонувати більш реалістичні та захоплюючі бойові сцени, а також більш складні та стратегічні битви. Це дозволяє створювати динамічні та непередбачувані ігрові ситуації, що тримають гравців у напрузі та викликають почуття задоволення від перемоги.

Слешери продовжують розвиватися, залучаючи нові технології, інноваційні механіки та створюючи захоплюючі історії. Вони залишаються важливою частиною ігрової індустрії, пропонуючи гравцям унікальний досвід, що поєднує інтенсивні бої, глибокий сюжет і вражаючу графіку.

Аналізуючи певний жанр ігор, важливо виділити та дослідити певні, вже існуючі проекти на ринку. Дана практика надає можливість виділити певні тенденції та дізнатись за які нововведення та/або особливості цінуються ці проекти і перейняти позитивний досвід.

DeadCells: це інді-гра в жанрі rogue-like metroidvania, розроблена та випущена французькою студією Motion Twin у 2018 році. Вона поєднує в собі елементи кількох жанрів, включаючи hack and slash, rogue-like та metroidvania, і стала однією з найуспішніших інді-ігор свого часу. Dead Cells є чудовим прикладом інноваційного підходу до створення гри, що поєднує динамічні бої, цікавий геймплей та високий рівень складності. Кадр з ігрового процесу «DeadCells» приведено на рисунку 1.2:

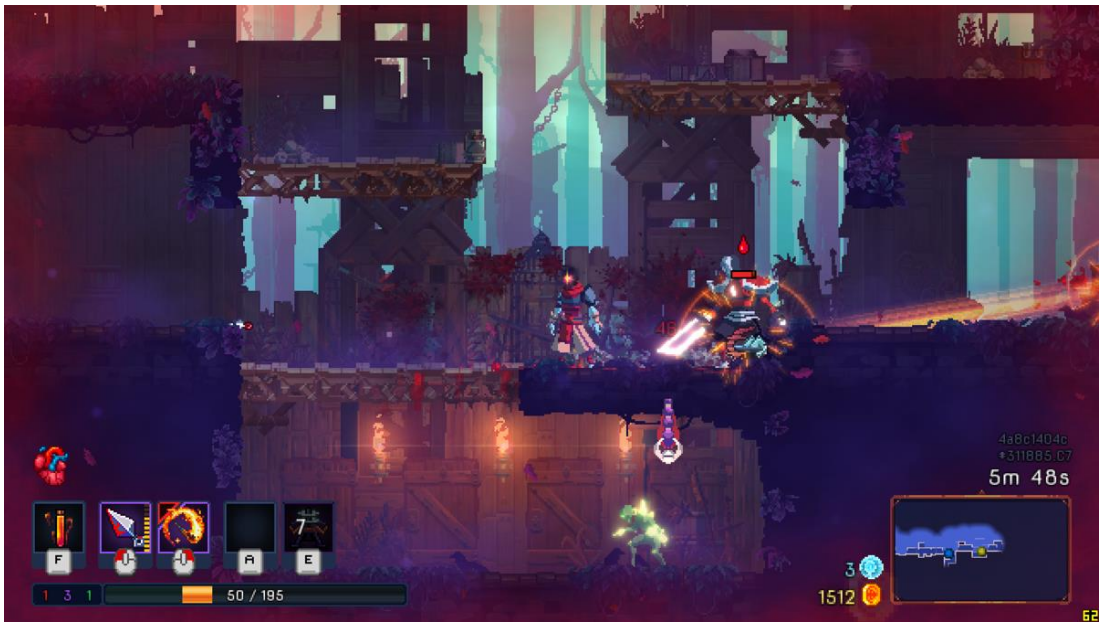


Рисунок 1.2. Кадр гри «DeadCells»

Бойова система, яка визначає цей проект як слешер, будується на різноманітному арсеналі зброї, яка доступна гравцеві. Це включає мечі, кинджали, молоти, списи, луки, арбалети та магічні артефакти. Кожен тип зброї має свої переваги та недоліки, що дозволяє гравцям експериментувати і знайти оптимальний стиль гри. Також крім основної зброї, гравці можуть використовувати вторинні предмети, такі як гранати, пастки, щити та турелі. Це додає глибини бою та надає гравцеві більшу свободу вибору, дозволяючи йому вигадувати та використовувати різні тактики для проходження кімнати на його дорозі.

Важливими елементами ігри, які слід виділити – це пастки і статус ефекти. Вони, не дивлячись на свою простоту, додають глибини і різноманітності ігровому процесу.

Пастки часто представлені у вигляді шипів, які час від часу розташовуються посеред рівня, заставляючи гравця пересуватись обережніше, а інколи вони взагалі становлять цілу кімнату, акцентуючи більше увагу на платформерну складову гри.

Статус-ефекти, які накладаються на гравця або ворогів під час гри, можуть ускладнити або полегшити проходження локації. Так, прикладом позитивного статус-ефекту буде прискорення, яке гравець отримує після перемоги над певною

кількістю ворогів, за короткий відрізок часу, а негативного – кровотеча, може накладатись як на гравця так і на ворогів, залежно від ситуації в грі.

Katana ZERO: це стильний двовимірний екшн-платформер з елементами слешеру, розроблений студією Askiisoft та випущений Devolver Digital у 2019 році. Гра виділяється своєю стилістикою, інтенсивними боями та атмосферною музикою. На рисунку 1.3 зображено кадр з ігрового процесу «Katana ZERO»:

Рисунок 1.3. Кадр гри «Katana ZERO»

Ігровий процес даного проекту відрізняється дуже швидким темпом. Однією з особливостей гри є відсутність смужки здоров'я в головного героя. Це додає інтенсивності боям, яка лише підкреслюється грамотним дизайном локацій, вимагають від гравця уважності та швидкої реакції.

Іншою важливою особливістю гри є механіка уповільнення часу, яку гравець може використовувати протягом гри в обмеженій кількості. Цей елемент працює на облегшення і поглиблення ігрового процесу, допомагаючи гравцеві проходити занадто складні відрізки гри, але заставляючи обмислювати і стратегічно використовувати цей цінний та обмежений ресурс.

Hades: це екшн-гра у жанрі rogue-like з видом зверху, розроблена та випущена студією Supergiant Games у 2020 році. «Hades» є одним із нечисленних прикладів використання грецької міфології у грі в якості сеттінгу, що лише додає цікавості і унікальності проекту.

На відміну від попередньо зазначених проектів, «Hades» пропонує менший арсенал, та більш дружній до нового гравця ігровий процес. На початку, з і без цього невеликого арсеналу, доступна лише його частина, яка відкривається протягом гри. Приклад проходження кімнати в «Hades» зображено на рисунку 1.4:

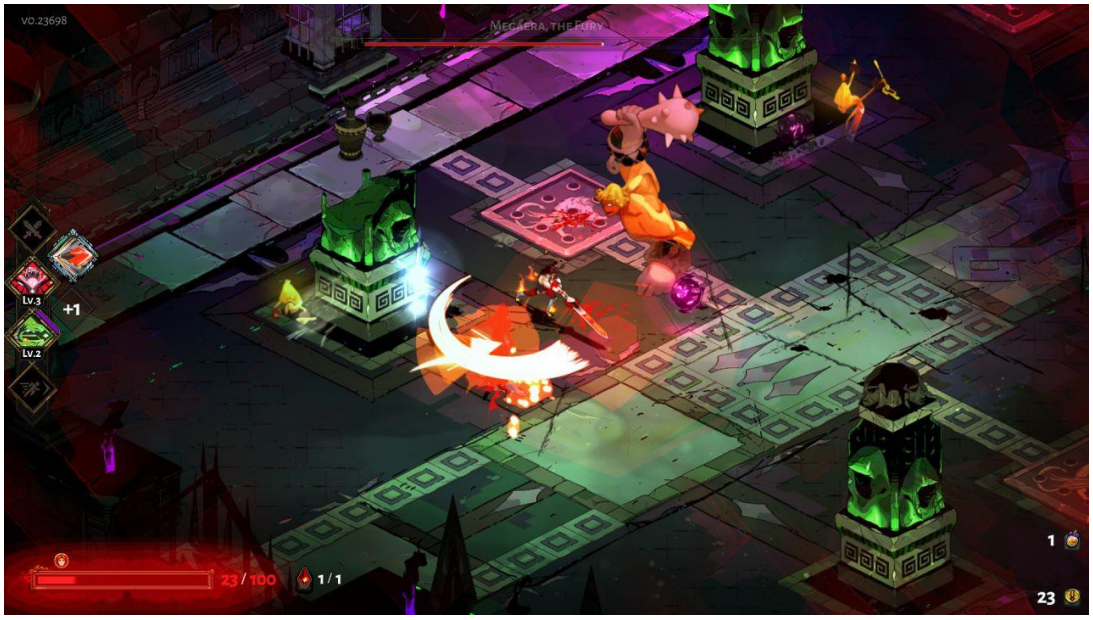


Рисунок 1.4. Кадр гри «Hades»

Там де «DeadCells» і «Katana Zero» роблять ставку на різноманітність опцій в гравця і в загалом мають більший акцент на мехінки пов’язані з головним героєм, «Hades» додає до різноманітності ворогів та пасток на локаціях, перевіряючи уважність і реакцію гравця.

В «Hades» через велику кількість різних типів ворогів різні кімнати здаються цікавішими, а різні локації, які герой відвідує протягом проходження гри, більш унікальними. Кожна наступна локація має свій унікальний перелік ворогів, які відрізняються більшою складністю від попередніх, та, нерідко, мають унікальні механіки, що відрізняють їх.

Висновок:

Аналізуючи різні проекти в жанрі «слешер» або з елементами слешеру, можна побачити що в своїй основі вони дуже схожі. Вони надають різноманітний арсенал гравцеві та локацію повну ворогів. Головним чином, вони відрізняються в інших, дрібніших деталях і механіках, придумуючи безліч різноманітних

способів прикрасити простий і прямолінійний ігровий процес, що робиться через уважну роботу над основними механіками, такими як рух і бій, задля більш плавного та приємного ігрового процесу та/або впровадженням додаткових механік, які поглиблюють ігровий процес.

1.2 Аналітичний огляд програмних рушіїв з умовно безкоштовним доступом

1.2.1 Аналіз ігрового рушію Unity

Unity – це потужний ігровий рушій, розроблений Unity Technologies, який став одним із найбільш популярних інструментів для створення ігор, завдяки своїй доступності та широкому функціоналу.

Unity підтримує безліч платформ, включаючи Windows, macOS, Linux, iOS, Android, консолі (PS4, PS5, Xbox One, Xbox Series X/S, Nintendo Switch), а також веб-платформи за допомогою WebGL. На рисунку 1.5 приведений приклад роботи в Unity:

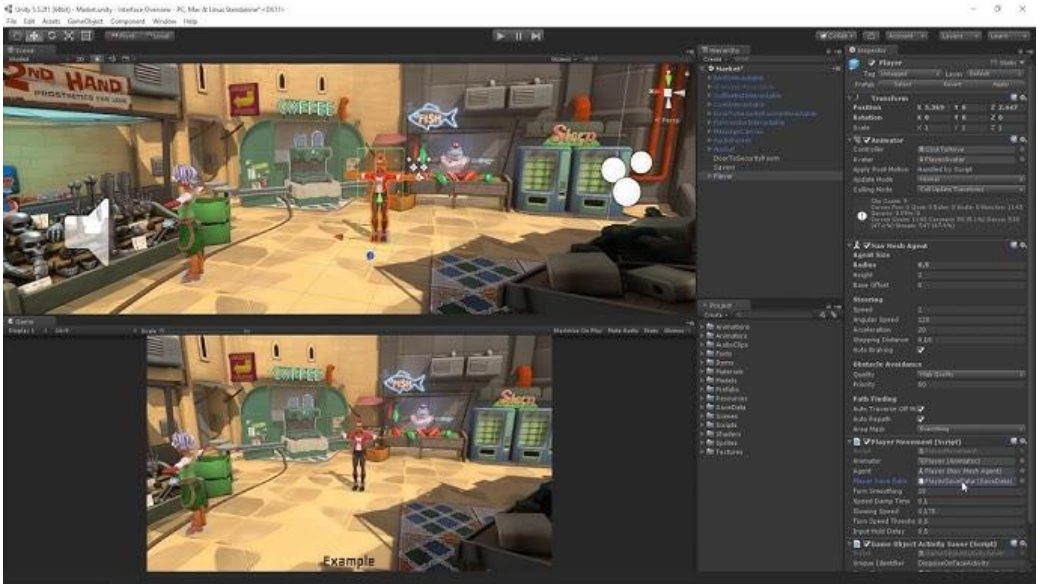


Рисунок 1.5. Приклад роботи в Unity

Рушій підтримує 2D і 3D графіку, має потужні інструменти для створення та налаштування матеріалів, шейдерів та візуальних ефектів, що дозволяє створювати як і прості, так і візуально прекрасні проекти, які нічим не поступаються конкурентам.

Однією з важливих переваг Unity є його велика спільнота користувачів, що забезпечує велику кількість освітніх матеріалів у вільному доступі стосовно цього рушія, що забезпечує більш легке вивчення його інструментів та особливостей.

1.2.2 Аналіз ігрового рушію Unreal Engine

Unreal Engine – це потужний ігровий рушій, розроблений Epic Games, який відомий своїми високоякісними графічними можливостями та широким спектром інструментів для розробників.

Підтримуючи більшість сучасних ігрових платформ, Unreal Engine надає багатий та гнучкий інструментарій для розробки ігор. На рисунку 1.6 приведений приклад роботи в Unreal Engine:

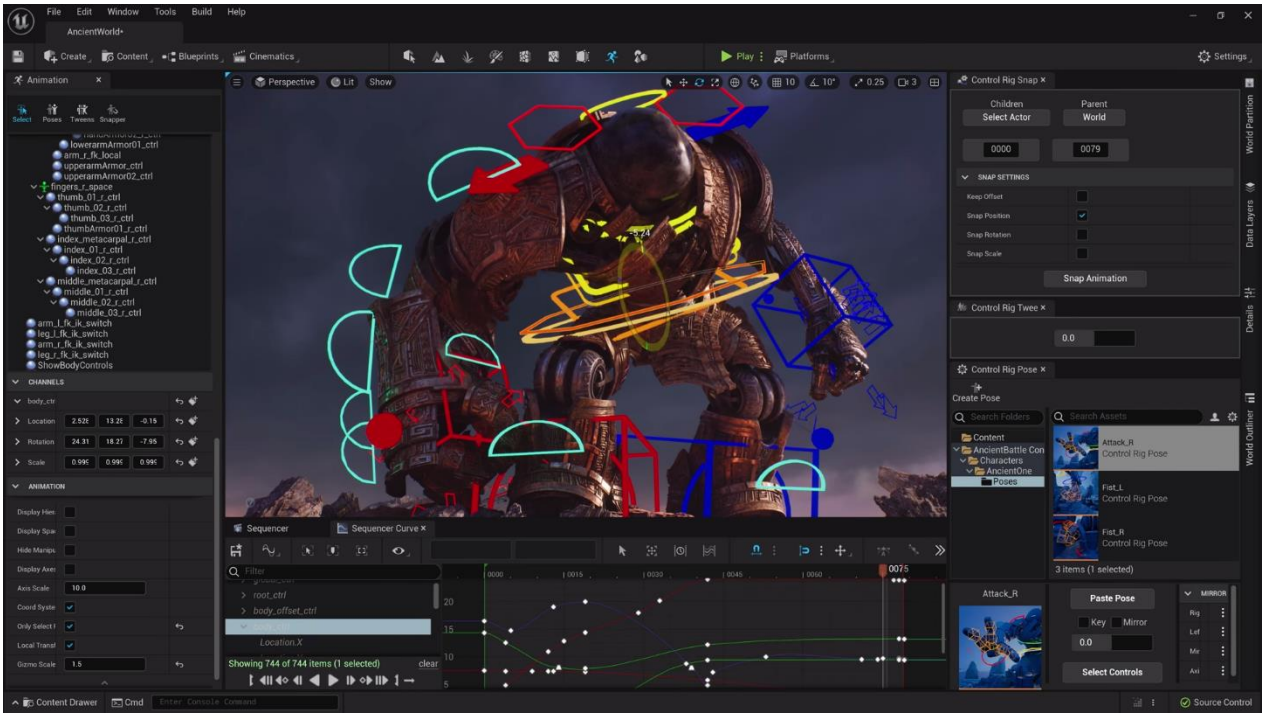


Рисунок 1.6. Приклад роботи в Unreal Engine

«Із коробки» цей рушій дуже дружелюбний до нових користувачів. Він має свою візуальну систему створення скриптів «Blueprints», яка дозволяє створювати ігрову логіку для проекту, дозволяючи розробнику не заглиблюватися в код без необхідності. Ця функція корисна як для нових розробників, які ще не поспішають писати код самостійно, так і для досвідчених кодерів, яким ця система допомагає прототипізувати нові функції, для тестування їх роботи на практиці.

Маючи не менше освітньої документації і форумів, ніж Unity – Unreal Engine

є дуже дружелюбним для новачків, та водночас має великий потенціал для досвідчених розробників, дозволяючи створювати не просто дивовижно красиві і атмосферні проекти, але й робити їх оптимізованими, використовуючи в якості своєї основної мови програмування C++.

1.2.3 Аналіз ігрового рушію Godot Engine

Godot Engine – це потужний та безкоштовний ігровий рушій з відкритим вихідним кодом, який активно розвивається та підтримується командою ентузіастів з усього світу. Він відзначається своєю гнучкістю та багатофункціональністю, що робить його привабливим вибором для інді-розробників та невеликих студій. На рисунку 1.7 приведений приклад роботи над проектом в Godot Engine:

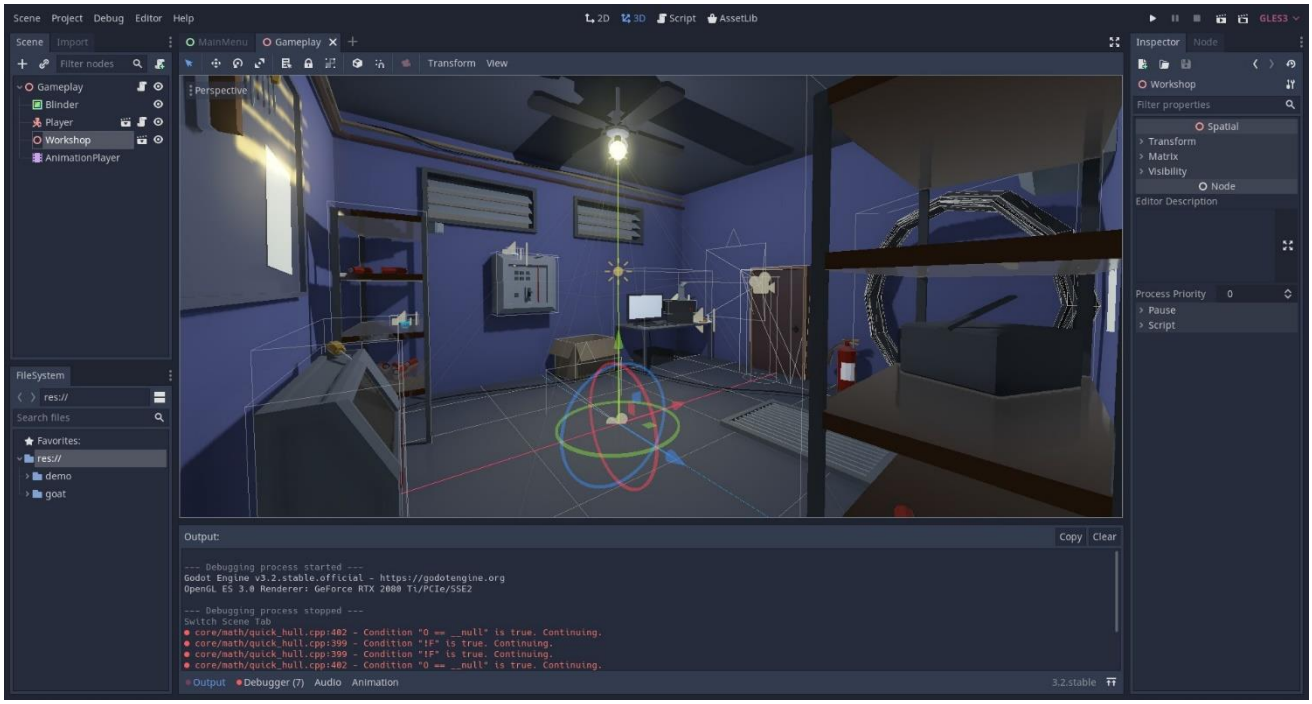


Рисунок 1.7. Приклад роботи в Godot Engine

Godot Engine підтримує розробку проектів під такі платформи як: Windows, macOS, Linux, Android, iOS та веб-платформи через HTML5. Цей рушій підтримує розробку проектів з 2D та 3D графікою, надаючи весь інструментарій, який може знадобиться рядовому розробнику: У 2D режимі пропонуються інструменти для роботи зі спрайтами, анімаціями та фізикою, що дозволяє створювати високоякісні двовимірні ігри. У 3D режимі рушій підтримує роботу з матеріалами,

шейдерами, світлом та ефектами, що дає можливість створювати красиві тривимірні сцени.

В якості своєї основної мови програмування Godot використовує власну мову програмування – GDScript, проте він надає можливість писати скрипи і на таких мовах як C# та VisualScript, що робить його доступним для більшого спектру різних розробників.

1.2.4 Аналіз ігрового рушію CryEngine

CryEngine – це потужний ігровий рушій, розроблений німецькою компанією Crytek. Він відомий своїми можливостями для створення фотореалістичної графіки та великих відкритих світів. CryEngine є одним із найбільш технологічно просунутих рушіїв на ринку на сьогоднішній день.

CryEngine надає можливість розробляти проекти для багатьох популярних ігрових платформ, таких як Windows, PlayStation, Xbox, а також підтримує розробку проектів під VR/AR пристрої.

CryEngine відомий своїми можливостями для створення фотореалістичної графіки. Рушій забезпечує потужний рендеринг у реальному часі, підтримку сучасних технологій освітлення, шейдерів та пост-ефектів, що дозволяє створювати вражаючі візуальні ефекти та реалістичні сцени.

Як приклад проекту, розробленого на цьому рушії, можна відзначити серію ігор «Crysis», яка була розроблена на різних версіях CryEngine, та починаючи с першої ігри в серії демонструвала потужні графічні можливості цього рушія. З його допомогою розробники можуть використовувати передові технології рендерингу, освітлення та матеріалів для створення захоплюючих візуальних ефектів та реалістичних середовищ, а серію ігор «Crysis» до цих пір пам’ятають як видовищні ігри, що випередили стандарти свого часу. На рисунках 1.8 та 1.9 відповідно можна побачити роботу над проектом в CryEngine та кадр з ігрового процесу третьої частини серії «Crysis»:

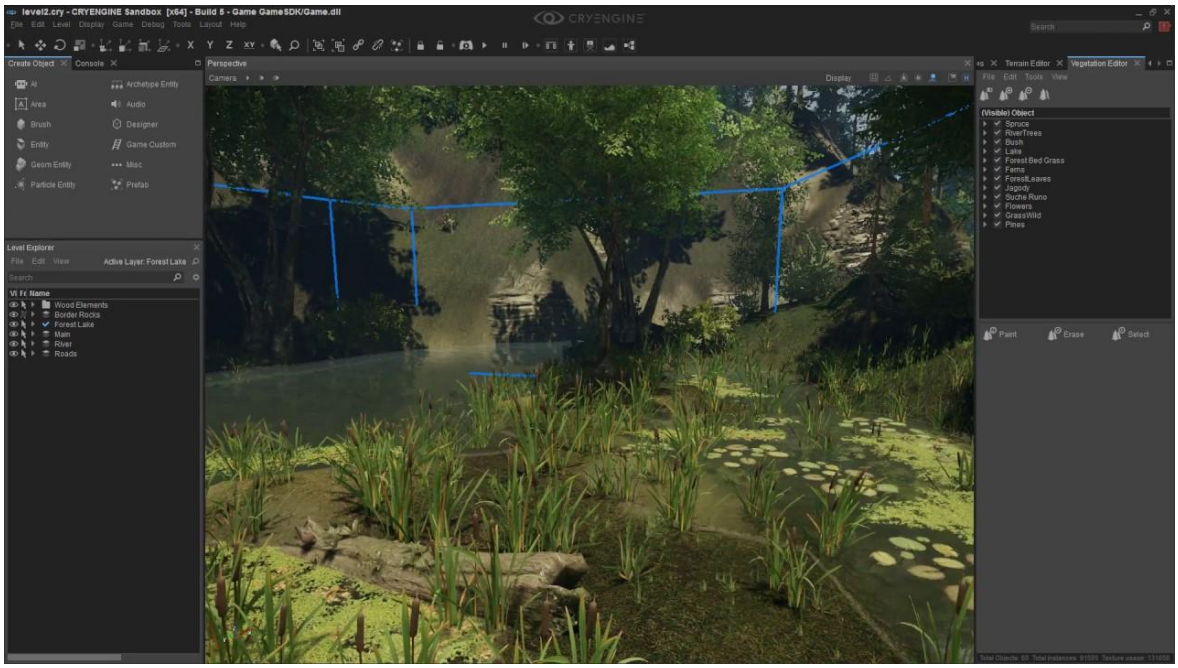


Рисунок 1.8. Приклад роботи в CryEngine

Рисунок 1.9. Кадр гри «Crysis 3»

Також цей рушій використовує Lua для скриптів та C++ для високорівневого програмування, що надає розробникам велику гнучкість у створенні ігрової логіки та механік.

Його потужні інструменти, передові технології рендерингу та підтримка різних платформ роблять його привабливим як для інді-розробників, так і для великих студій.

1.3 Формування концепту ігрового процесу 2D-гри в жанрі слешер

Ігровий процес – основний елемент будь якої гри. Це той елемент, який відрізняє один проект від інших та виділяє ігри як такі від інших форм електронних розваг.

В основі будь якого ігрового процесу стоїть ігровий цикл(«gameloop»). Ігровий цикл – це низка дій, які протягом гри виконує гравець. Ігровий процес будь-якої гри можна, зрештою, звести до короткого опису активностей, якою б комплексною гра не здавалась.

В якості прикладу можна взяти «DeadCells» від MotionTwin. Якщо розбити ігровий цикл цього проекту на пункти, то він буде виглядати наступним чином:

1. Гравець заходить на локацію --> 2. В процесі дослідження, гравець зачищає локацію --> 3. Під час переходу до наступної локації гравець отримує підсилення та відновлює здоров’я --> повернення на початок циклу.

Все інше, за рамками цього ігрового циклу, лише прикрашає ігровий процес, додаючи глибину та різносторонність грі.

Тепер, знаючи з чого треба починати, треба сформувати ігровий цикл для цього проекту.

Ігровий цикл даного проекту, при впровадженні всіх необхідних ігрових механік має виглядати наступним чином: гравець, з’являючись на локації, відбиває хвилі противників, після чого чекатиме на появу наступної хвилі, а далі цикл повторюється. Коли гравець не зможе відбити чергову хвилю – він програє, після чого побачить рахунок очок, які він заробив протягом гри, та зможе перезапустити гру або вийти з неї. Такий простий цикл ігрового процесу дозволятиме зосередити зусилля на впровадженні кореневих механік, необхідних для його реалізації

На рисунку 1.10 можна побачити схему розробленого ігрового циклу:

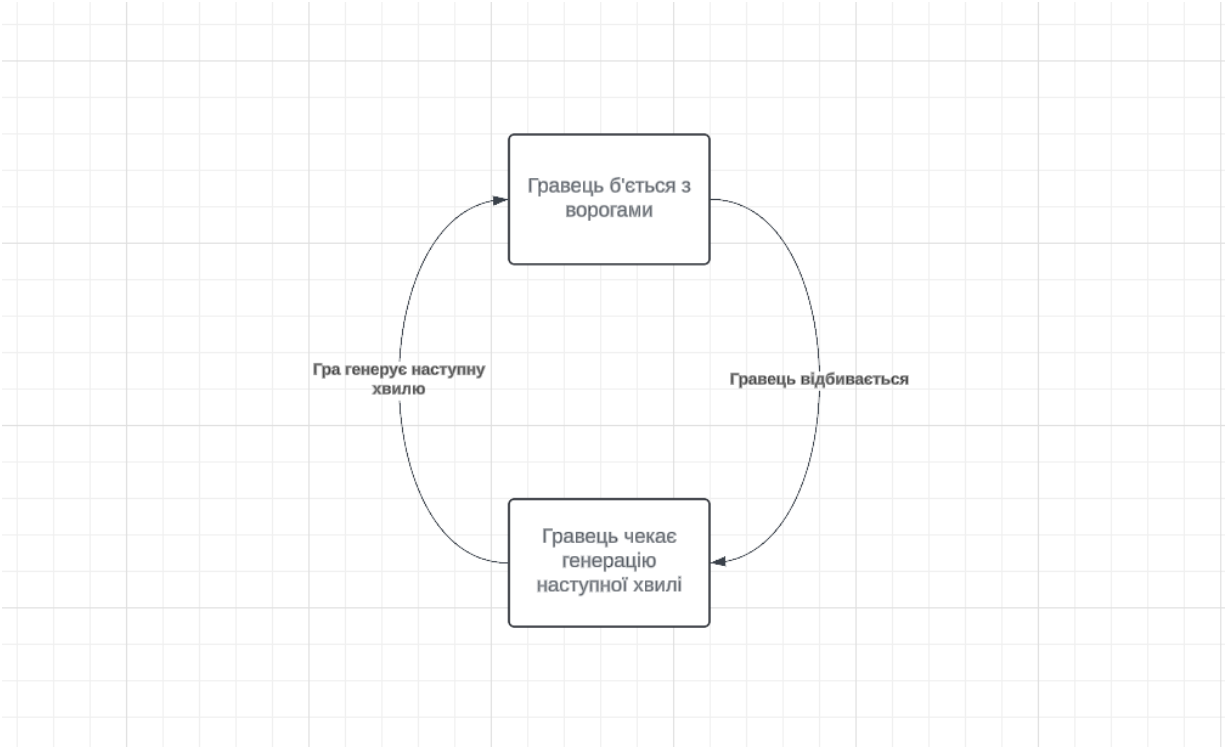


Рисунок 1.10. Ігровий цикл розроблюваного проекту

1.4 Проектування основних елементів ігрового процесу та принципи їх роботи

Навіть якщо гра здається візуально і технічно простою – вона легко розбивається на багато маленьких елементів, в реалізації яких є свої нюанси, деталі та складнощі.

Збираючи свій проект, слід обмислити та спланувати з яких механік він буде складатись. Існує безліч різних механік і елементів вже реалізованих в інших проектах, які можна запозичувати або ними надихатись для створення чогось нового. Ці механіки можна розподілити на розповсюдженні загальноприйняті та притаманні лише певним іграм, залежно від їх ігрового процесу, який вони намагаються реалізувати.

Так, якщо рух в межах ігрової сцени це одна з най-, якщо не найрозповсюджена механіка, притаманна багатьом проектам, починаючи ще з ранніх років розвитку ігор як таких, то є більш нішеві та складні системі імплементовані лише в деякі проекти. Наприклад: NEMESIS, із серії ігор Middle-earth: Shadow of Mordor та Shadow of War: ця унікальна система дозволяє ворогам

запам'ятовувати попередні зустрічі з гравцем, розвиватися та зростати у ранзі. Вороги можуть стати сильнішими, змінювати свій зовнішній вигляд і навіть мати власні особисті мотиви щодо гравця, що сприяє зануренню в ігровий світ.

Одна з фундаментальних механік, яку слід запровадити це рух персонажа, яким гравець керуватиме. Жанр «2D-слешер» має на увазі, що проект матиме двовимірну графіку. Відштовхуючись від цього, є два основних способи реалізації руху:

1. Рух з видом збоку («Side-Scrolling»): Гравець бачить гру з бокової перспективи, тобто всі об'єкти та персонажі відображаються збоку. Це дозволяє бачити персонажів у профіль і спостерігати за рухами по горизонтальній площині. Рух зазвичай обмежений двома напрямками: ліворуч і праворуч, що забезпечує більш лінійну перспективу та ігровий процес.

2. Рух з видом зверху (Top-Down): Гравець бачить гру зверху, наче дивлячись вниз на ігровий світ з висоти пташиного польоту. Це дозволяє бачити великий простір навколо персонажа та краще орієнтуватися в середовищі. При такій системі рух зазвичай вільний по всім чотирьом напрямам, включаючи діагональне переміщення.

Після оцінки позитивних та негативних сторін обох варіантів, в даному проекті буде реалізована система руху з видом зверху.

Також як персонаж, яким управляє гравець, так і його вороги повинні реагувати на атаки один одного. Можна реалізувати систему здоров’я, яка буде відстежувати заздалегідь приписану кількість очок здоров’я. Для реалізації системи здоров’я, буде створено універсальний скріпт, який може бути прикріплений як до гравця, так і до ворогів, для запобігання написання різного коду для виконання подібних задач.

Даний скрипт матиме дві змінні: одна зберігатиме максимальний рівень здоров’я, а інша динамічний рівень здоров’я, який змінюватиметься протягом гри. Цей скрипт відстежуватиме динамічний рівень здоров’я, видаляючи зі сцени носія скрипту, якщо він падає нижче 1 та прирівнюватиме рівень здоров’я до максимального, якщо при додаванні здоров’я сума буде виходити більша за

максимальне виставлене значення.

Також саме скрипт здоров’я буде зберігати функції лікування, тобто поповнення здоров’я, та нанесення урону, яка, відповідно, відніматиме здоров’я від носія скрипту в певному обсязі.

Для героя і ворогів, які будуть йому протистояти, будуть створені коллайдери. Коллайдер, в свою чергу, - це компонент, який визначає фізичну форму об'єкта для системи фізики. Він використовується для виявлення зіткнень між об'єктами у грі та взаємодії з іншими фізичними компонентами, такими як Rigidbody. Коллайдери можуть мати різні форми (сфери, куби, капсулі тощо) і можуть бути як тригерними(використовуються для виявлення зіткнень без фізичних реакцій) так і нетригерними(використовуються для реальних фізичних взаємодій). Коллайдери про які будуть додані до персонажів на сцені – саме тригерні і вони будуть відстежувати контакт один з одним. При контакті, ці коллайдери будуть викликати функцію, в скрипті здоров’я атакованого, яка знижатиме його рівень здоров’я відповідно до нанесеного урону. На рисунку 1.11 зображені прототипи ворогів і героя:

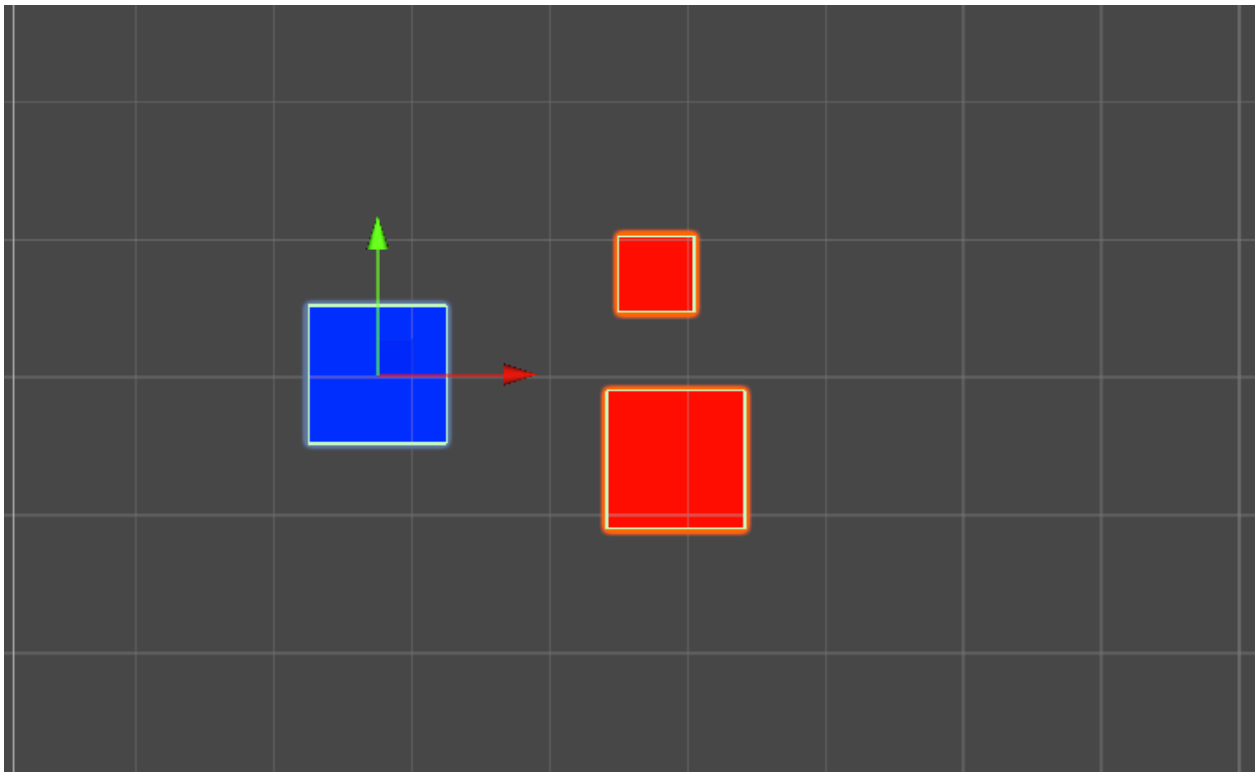


Рисунок 1.11. Коллайдери на прототипах героя і ворогів

Надалі треба розробити безпосередньо ворогів, які повинні протистояти гравцеві.

Буде всього два типи ворогів: маленький, та великий. В них буде відрізнятись такі параметри як швидкість, урон та здоров'я. Більші за розміром вороги матимуть більше здоров'я, та наноситимуть більший урон гравцеві, проте будуть повільнішими за маленьких побратимів.

Також ці вороги повинні якимось чином з'являтись на сцені. Для цього на карті повинні бути розташовані точки їх появи –спавнери(від англ. «spawn» - «породжувати»). Далі, після появи на сцені, вони слідуватимуть за гравцем від цієї точки, наносячи йому урон при контакті, до тих пір, поки в гравця не закінчиться здоров'я, після чого гра закінчиться.

За раз повинна буде з'являтись певна кількість ворогів. Тобто гра повинна відстежувати кількість ворогів на карті, і не генерувати нових, коли потрібна кількість на «раунд» вже є на сцені, та не генерувати нову групу, поки гравець не справився зі старою.

Для реалізації подібної системи появі одного ворога буде приписана певна ціна в очках. В свою чергу в гри на кожний «раунд» буде певна кількість цих очок – «тікетів»(ticket). Рахунок тікетів використовуватиметься для генерації нових ворогів, віднімаючи їх від загальної суми за кожного згенерованого ворога. На рисунку 1.12 зображений відрізок коду, відповідальний за генерацію тікетів:

```
    }

    Ссылок: 2
    private int Tickets(int rounds) {

        int result = rounds + 10;
        Debug.Log("Зарандомив " + result + " тікетів");

        return result;

    }

    Ссылок: 2
    private void EnemyTypesGenerator(int tickets) {
```

Рисунок 1.12. Система тікетів

Починаючи з першого раунда – гра матиме тікетів за формулою *n+10*, де «n» - це рахунок раундів. Це сприяє поступовому зростанню складності з кожним новим раундом.

Різні типи ворогів також будуть відрізнятись за ціною тікетів за генерацію. Великі – будуть коштувати 2 тікети, маленькі – 1 тікет. Ще гра повинна перевіряти наявність достатньої кількості тікетів при генерації великого ворога. У випадку нестачі тікетів гра генеруватиме замість нього – маленького, після чого генерація припиниться через те що значення тікетів опустилося до нулю. Більше тікетів не з’явиться до того як згенерована група ворогів не буде подолана гравцем.

На виході повинен бути персонаж, яким управлятиме гравець, точки генерації противників, де останні будуть з’являтись групами, а їх чисельність контролюється грою та збільшує з часом.

1.5 Реалізація основних елементів ігрового процесу. Ігрові механіки

1.5.1 Реалізація ігрової механіки руху:

Для імплементації руху з видом зверху у проект, потрібно створити і додати на сцену об’єкт, яким буде управляти гравець. На цьому етапі, гра матиме графіку зіставлену з примітивів, котрі згодом будуть замінені повноцінними спрайтами.

Для початку треба створити пустий ігровий об’єкт на сцені. Викликавши контекстне меню правим кліком в ієрархії об’єктів – створюємо пустий об’єкт, натиснувши «create empty».

Після цього на сцені з’являється невидимий ігровий об’єкт, з яким можна взаємодіяти і змінювати його, додаючи нові елементи.

Тепер, для того щоб ігровий об’єкт було видно, потрібно додати до нього спрайт, модель або, як в цьому випадку, геометричний примітив, який буде тимчасово заміняти повноцінний, двовимірний спрайт.

На рисунках 1.13 та 1.14 можна побачити створення пустого ігрового об’єкту та модифікацію прикріпленого до нього примітива-куба відповідно:

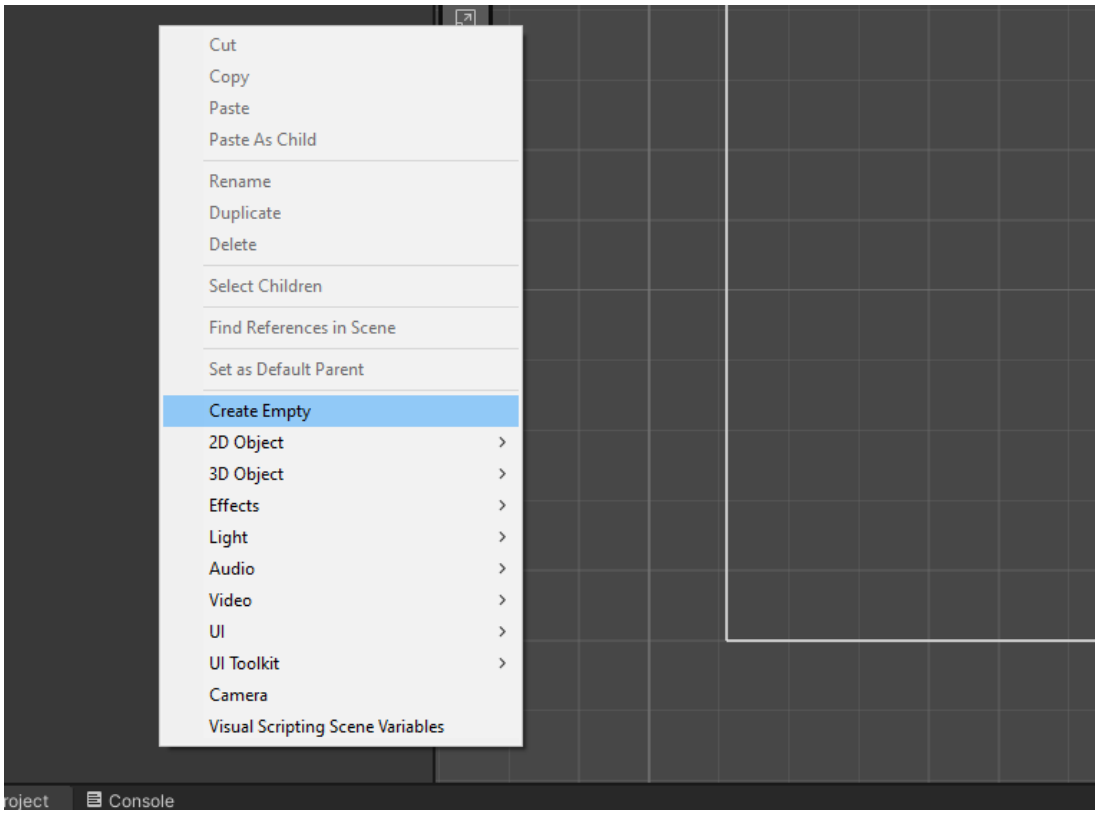


Рисунок 1.13. Створення пустого об’єкту на сцені

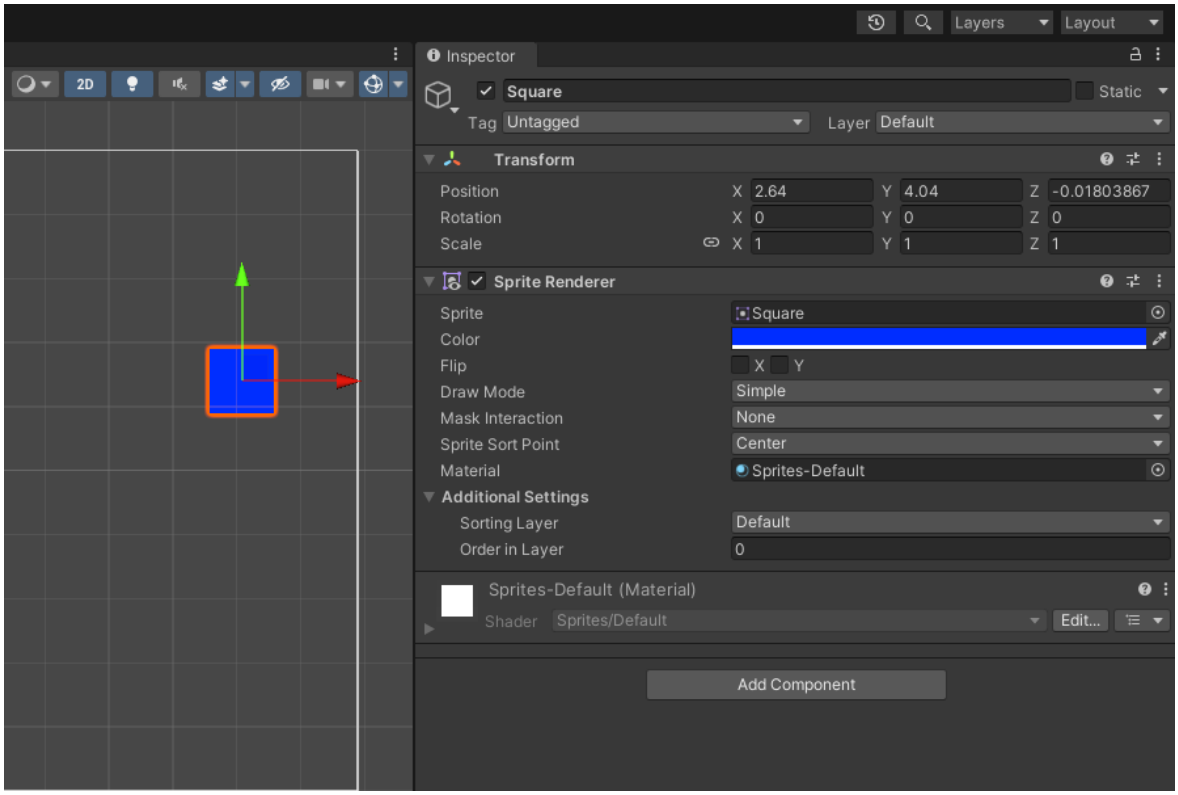


Рисунок 1.14. Модифікація примітива-квадрата

Створивши таким самим чином примітив-квадрат – додаємо його до створеного раніше пустого ігрового об’єкту через ієрархію. Тепер, щоб надалі

було простіше відрізняти майбутнього героя від інших об’єктів на сцені, модифікуючи елемент квадрату під назвою «sprite renderer» змінюємо його колір на синій.

Після цього об’єкт можна перейменувати в ієрархії на «Player», та, відповідно, змінити його однойменний тег в інспекторі елементів.

Таким чином було створено візуальне відображення об’єкту, яким управлятиме гравець, та до якого можна буде в майбутньому прикріпити всі необхідні елементи та легко їх налагодити. На рисунку 1.15 зображений результат внесених раніше модифікацій до ігрового об’єкту:

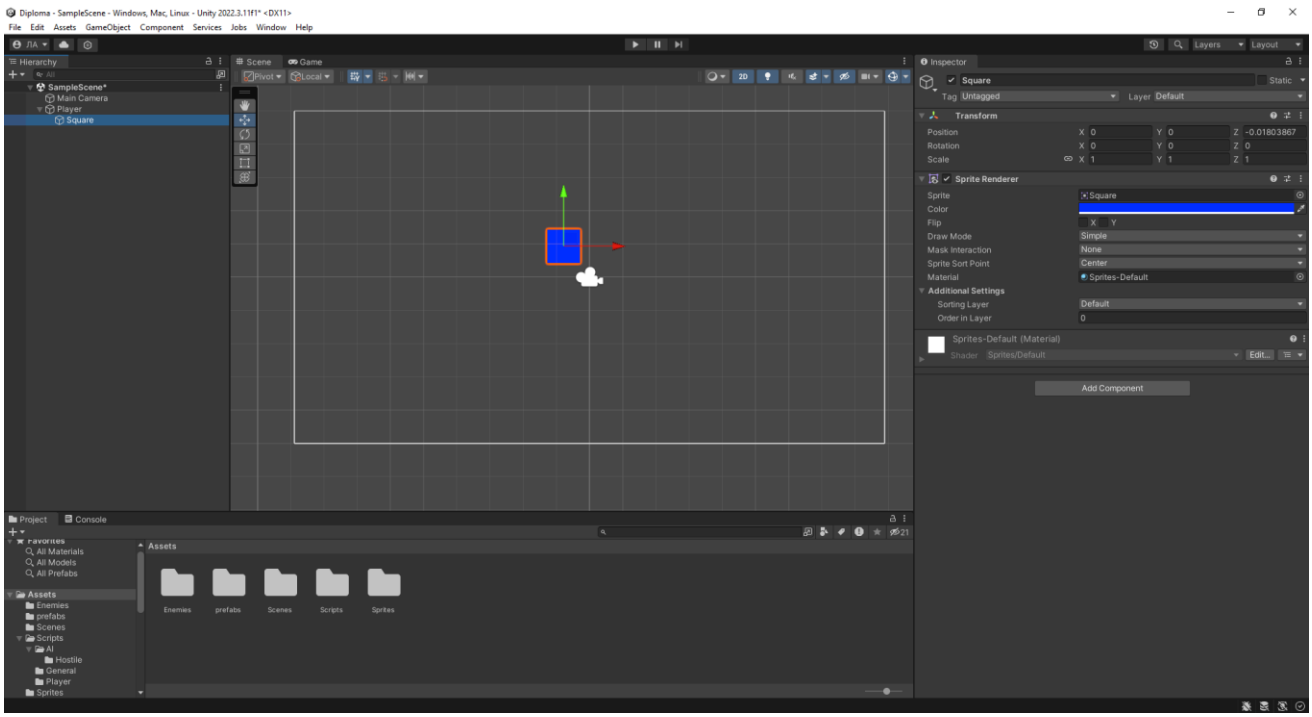


Рисунок 1.15. Результат модифікацій внесених до пустого ігрового об’єкту

Слід зауважити, що через те що примітив-куб прикріплений до пустого ігрового об’єкту – саме пустому об’єкту було змінено тег та назву в ієрархії на «Player» і для внесення будь яких візуальних змін до даного об’єкту – необхідно працювати з, в даному випадку, примітивом.

Тепер для того щоб гравець міг управляти створеним на сцені об’єктом – необхідно створити новий C# скрипт, який за це відповідатиме.

Перед створенням і роботи зі скриптом – в нижньому полі «Assets», визвавши контекстне меню, створюємо папку для скриптів «Scripts», та в ній

створюємо ще одну папку «Player», куди будуть вкладатись всі скрипти пов’язані із гравцем. Важливо структуровано розкладати всі вихідні матеріали, для легкої орієнтації в проекті та швидкого доступу до необхідних елементів проекту.

Після проведення підготовчих робот – в директорії Assets\Scripts\Player створюємо скрипт «PlayerCtrl», який надаватиме гравцю можливість управляти об’єктом «Player» на сцені. Результат створення нового скрипту можна побачити на рисунку 1.16:

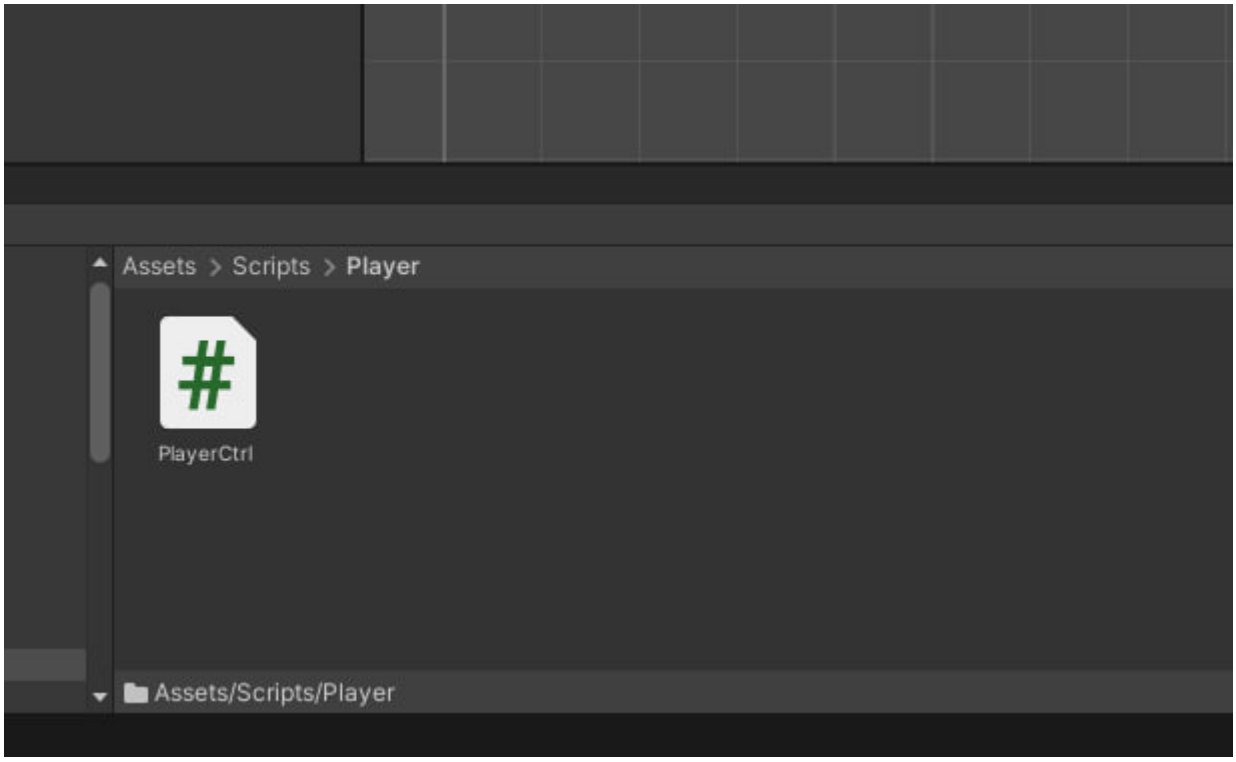


Рисунок 1.16. Створений скрипт «PlayerCtrl»

Тепер, коли був створений новий C# скрипт – треба відкрити його в редакторі коду і внести все необхідні зміни, для надання йому необхідного функціоналу.

```
public float movSpeed;

Rigidbody2D rb;
private Vector2 axisMovement;
void Start()
{
   rb = GetComponent<Rigidbody2D>(); //RigBody for movement
}
void Update()
{
   axisMovement.x = Input.GetAxisRaw("Horizontal");
```

```
        axisMovement.y = Input.GetAxisRaw("Vertical");
    }
    private void FixedUpdate()
    {
        Move();
    }
```

Даний скрипт повинен легко налагоджуватись під час роботи з проектом з самого Unity. Для цього в першій половині коду скрипту «PlayerCtrl», приведеного вище, при оголошенні необхідних для роботи змінних – змінна «movSpeed», яка відповідатиме за швидкість, з якою гравець рухатиметься по сцені, оголошена публічною(public). При використанні саме такого модифікатора, дана зміна, при прикріплені даного скрипту до об’єкту буде відображатись в інспекторі, в правій частині інтерфейсу Unity. Таким чином, дана зміна може бути легко відредагована під час тестування роботи скрипту.

Зміна «rb» - необхідна для того щоб скрипт міг побачити елемент «Rigidbody2D» прикріплений до носія скрипту, в даному випадку – об’єкт «Player».

Якщо звернути увагу, то можна побачити що «rb» прирівнюється до «GetComponent<Rigidbody2D>()». Даний елемент коду, який записаний в методі «Start», говорить про те, що на початку роботи скрипту він намагатиметься взяти елемент «Rigidbody2D» і записати його в зміну «rb», для подальшої взаємодії з ним в коді. Скрипт шукає даний елемент на носії сам по собі, тому після прикріплення скрипту до об’єкту – необхідно переконатись що елемент типу «Rigidbody2D» присутній на ньому та може бути знайдений при виконані цієї команди. В іншому випадку з’явиться некритична помилка, яка хоч і не закінчить роботу програми, проте буде говорити і свідчити про те що елемент не був знайдений і записаний в зміну, через що код не може виконувати свої функції.

Зміна «axisMovement» - необхідна для відстеження вводу зі сторони користувача за допомогою «axisMovement.x» та «axisMovement.y», які прописані в методі «Update».

«Update», в свою чергу, - це метод, який виконує код, який записаний в нього, кожний раз, коли рендериться кадр, або фрейм(«frame») гри, що корисно

при необхідності відстежувати ввід гравця.

Надалі прописаний метод «FixedUpdate». Як і «Update» Він виконує код, який в нього записаний, раз на короткий проміжок часу. Однак «Update», як було зазначено вище, виконує код кожний фрейм. Хоча це й корисно, наприклад, при відстеженні вводу зі сторони гравця – даний метод залежить від того скільки кадрів на секунду, або fps(від англ. «frames per second»), може видавати пристрій, на якому запущена програма. Тобто, ЕОМ з більшими обчислювальними потужностями може виконати один і той самий код більшу кількість разів за той самий проміжок часу що менш потужний пристрій. Це може викликати ряд проблем і в загалом негативно сказатись на ігровому досвіді користувача, якщо при різних показниках fps гра веде себе інакше, ніж заплановано розробником. На рисунку 1.17 зображений графік, який відображає різницю між «Update» «FixedUpdate»:

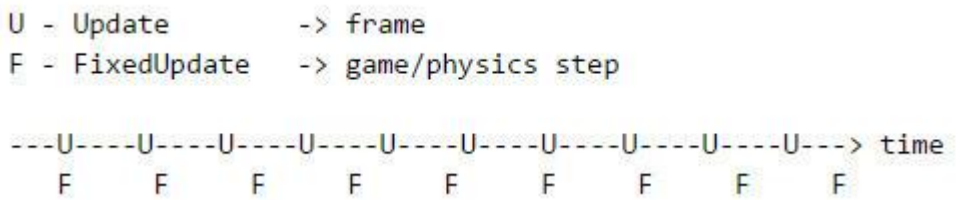


Рисунок 1.17. Графічне відображення різниці «Update» та «FixedUpdate»

Тож «FixedUpdate» виправляє вище зазначену проблему. Замість того щоб виконувати код кожний відрендерений фрейм – він виконує його кожний фіксований відрізок часу, який можна налаштовувати під потреби проекту. За замовчуванням «FixedUpdate» виконує код кожну 0.02 секунди, що дорівнює 50 виконуванням на секунду.

В методі «FixedUpdate» викликається інший метод – «Move», який прописан у другій половині коду, розібраного нижче.

```
private void Move() {
rb.velocity = axisMovement.normalized * movSpeed;
Check4Flipping();
   }
void Check4Flipping() {
bool movingLeft = axisMovement.x < 0;
bool movingRight = axisMovement.x > 0;
```

```
if (movingLeft) {
transform.localScale = new Vector3(-1f, transform.localScale.y);
}
if (movingRight) {
transform.localScale = new Vector3(1f, transform.localScale.y);
}
```

В другій половині коду прописані два важливі методи, які представляють собою головний функціонал цього скрипту: «Move» та «Check4Flipping».

Метод «Move» відповідає за безпосередньо переміщення об’єкту. В відрізку «rb.velocity = axisMovement.normalized * movSpeed;» швидкість «Rigidbody», який представляється змінною «rb», встановлюється за результатом виразу «axisMovement.normalized * movSpeed». «AxisMovement.normalized». В даному випадку, він нормалізує вектор напрямку руху, перетворюючи його на вектор одиничної довжини. Це дозволяє використовувати його як чистий напрямок без урахування початкової величини. Після нормалізації вектор множиться на змінну movSpeed, що визначає швидкість руху. Це означає, що об'єкт буде рухатися в напрямку axisMovement зі швидкістю, визначеною movSpeed.

Далі в методі «Move» визивається метод «Check4Flipping». Цей метод необхідний для зміни напрямку, в якому «дивиться» модель гравця. У випадку з примітивом-квадратом, який тимчасово виступає в ролі візуального відображення моделі гравця – це не видно, проте це дуже важливо, задля коректного відображення спрайту у майбутньому. Також це важливо з технічної точки зору, так як гравець буде атакувати за напрямом погляду об’єкту.

На початку методу визначається дві булеві змінні: «movingLeft» та «movingRight». Їх задача, відповідно назві, визначати в якому напрямку рухається об’єкт, до якого прикріплений скрипт і якщо ігровий об’єкт рухатиметься, наприклад, праворуч – то з цих двох змінних лише «movingRight» буде дорівнювати істині. Обидві змінні відстежують параметр координати «x» вектору axisMovement. Відповідно, якщо параметр координати «x» буде більший за нуль, то це означає що об’єкт рухається праворуч, і напроти – якщо «x» менший за нуль, то об’єкт рухається ліворуч.

Далі код перевіряє, яка зі змінних дорівнює істині. Якщо «movingLeft»

дорівнює істині – то скрипт змінює масштаб об'єкта по осі x на -1, не змінюючи змінні «y» і «z», що встановлює об’єкт «передньою» стороною спрямований вліво. Якщо напроти – «movingRight» дорівнює істині, то масштаб по осі x змінюється на 1, повертаючи об’єкт назад «лицем» вправо.

На виході – отримуємо скрипт, який зчитує ввід клавіш горизонтального і вертикального руху, якими є клавіші «W», «A», «S», «D», та рухатиме ігровий об’єкт з елементом «Rigidbody2D» за напрямом, відповідним вводу гравця; а також віддзеркалювати ігровий об’єкт залежно від напряму руху – вліво чи вправо.

Прикріпляємо цей скрипт до об’єкту «Player», перетягуючи його з поля «Assets» за допомогою системи «drag-and-drop», після чого також додаєм елемент «Rigidbody2D», який скрипт в процесі роботи, при ініціалізації, знайде самостійно і запише відповідне поле «rb» та зміг з ним взаємодіяти. На рисунку 1.18 можна побачити нові прикріплені елементи:

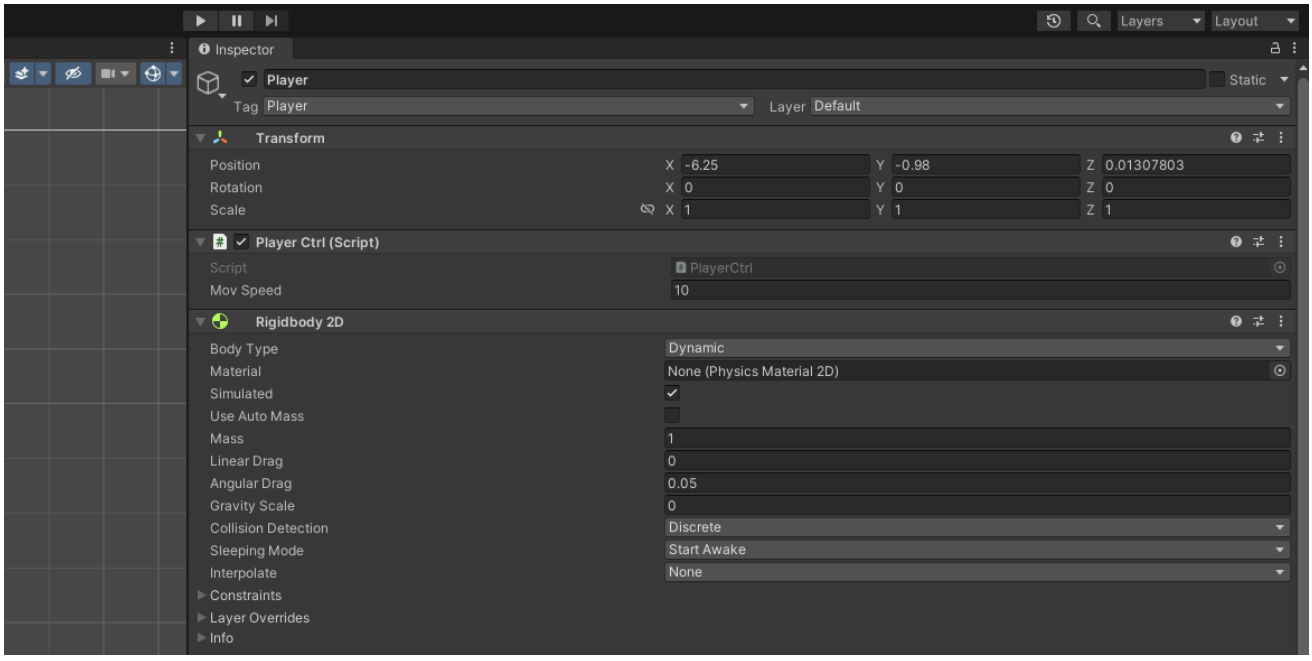


Рисунок 1.18. Прикріплені до об’єкту «Player» «PlayerCtrl» та «RigidBody2D»

1.5.2 Реалізація системи здоров’я:

Для того щоб гравець був спроможній наносити урон або його отримувати – потрібно створити скрипт для системи здоров’я. Для простоти імплементації скрипта – він буде універсальним як для гравця, так і для ворогів.

Перед створенням безпосередньо скрипту – необхідно створити папку «General» в директорії Assets/Scripts, продовжуючи уважне сортування вихідних матеріалів для проекту.

Після створення відповідної папки – створюємо в ній C# скрипт, для подальшої його модифікації і впровадження в проект. На рисунку 1.19 зображений даний процес:

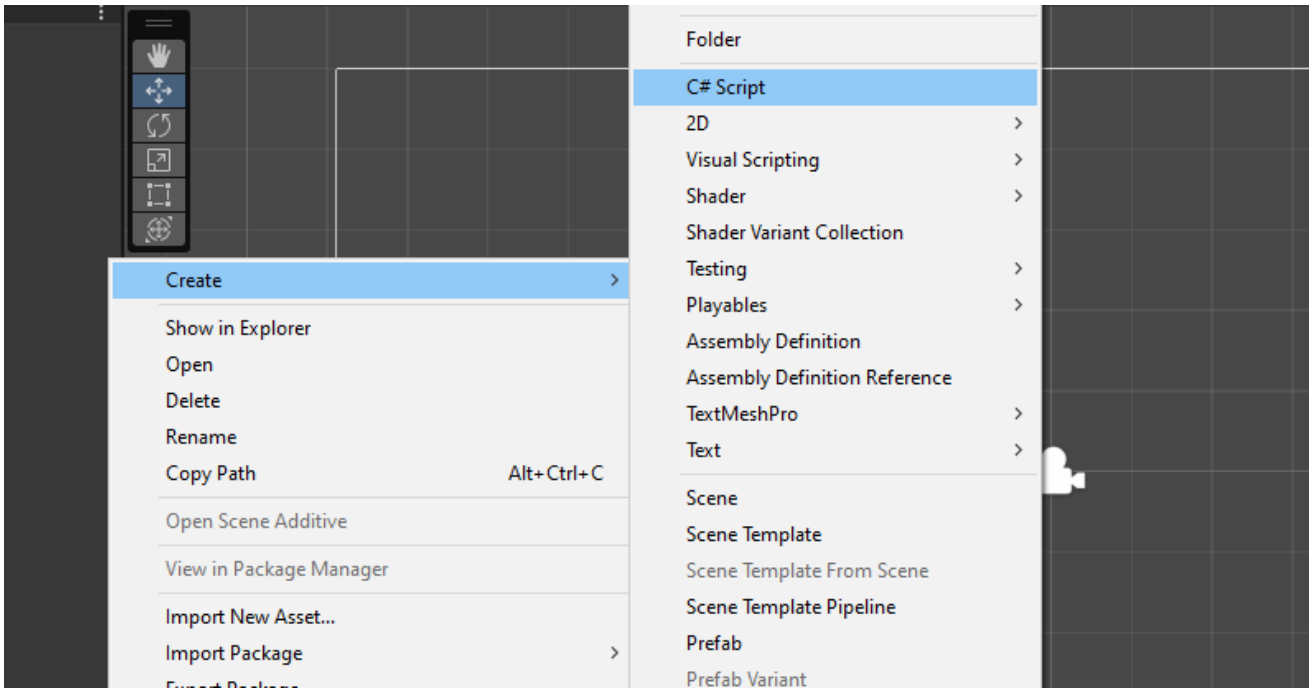


Рисунок 1.19. Створення скрипту

Для надання скрипту необхідного функціоналу – відкриваємо його, для внесення подальших змін.

На початку потрібно оголосити необхідні змінні, а саме «health» для динамічного рівня здоров’я і «MAX_HEALTH» для того щоб задавати максимальний рівень здоров’я.

Для повноцінного функціонування потрібно створити і прописати три методи, які будуть становити головний функціонал цього скрипту:

- Метод отримання урону;
- Метод поповнення здоров’я/лікування;
- Метод смерті.

На рисунку 1.20 зображена схема роботи системи здоров’я:

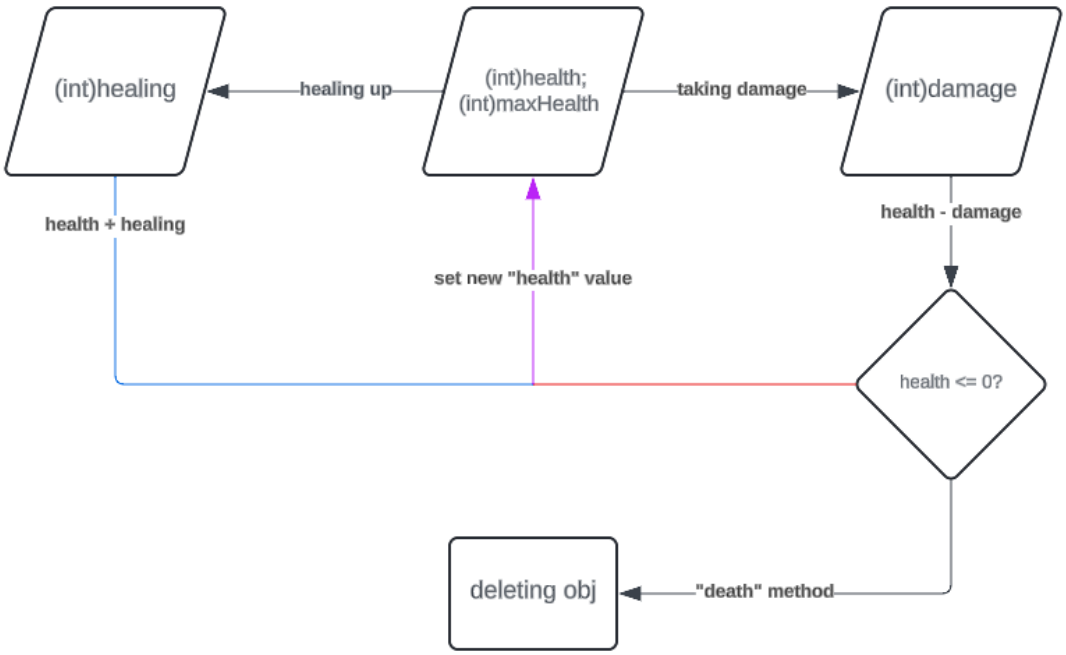


Рисунок 1.20 Схема роботи системи здоров’я

Носій скрипту, у випадку контакту з трігером, який наноситеме йому урон повинен мати метод, який прийматиме числову змінну, яка відображатиме кількість урону і відніматиме це значення від динамічного рівня здоров’я, який матиме носій на момент контакту.

```
public int health = 100;
public int MAX_HEALTH = 100;
public void Damage(int amount) {
if (amount > 0)
{
this.health -= amount;
}
else {
Debug.LogError("Can't have a negative damage");
}
}
public void Heal(int amount) {
if (amount > 0)
{
bool wouldBeOverMaxHealth = health + amount > MAX_HEALTH;
StartCoroutine(VissualIndicator(Color.green));
if (wouldBeOverMaxHealth)
{
```

```
health = MAX_HEALTH;
}
else {
this.health += amount;
}
}
else {
Debug.LogError("Can't have a negative healing");
}
}
private void Death() {
Debug.Log("I'm dead x_x");
Destroy(gameObject);
}
```

Так, в приведеному вище відрізку коду оголошуються необхідні змінні і методи «Damage», «Heal» та «Death».

В свою чергу, метод «Damage», після того як отримав на вхід певне число – перевіряє, чи воно менше нулю. Якщо число проходить перевірку метод віднімає від динамічного рівня здоров'я отримане на вході числове значення. В іншому випадку – метод виведе помилку у консоль. Схема даного методу зображена на рисунку 1.21:

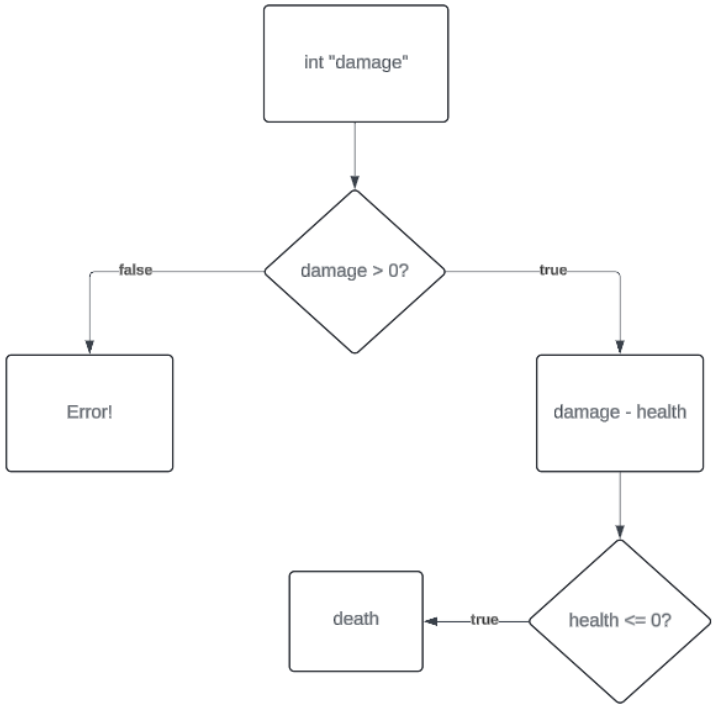


Рисунок 1.21. Схема роботи методу «Damage»

Тепер слід додати метод поповнення здоров’я. Він працюватиме схожим на метод отримання урону чином. На вхід метод «Heal» отримуватиме певне числове значення, яке, у випадку успішної перевірки, додаватиметься до рівня здоров’я. Проте, на відміну від методу «Damage», - «Heal» перед тим як додавати буде перевіряти отримане на вхід значення, і якщо при додаванні воно перебільшуватиме значення максимального здоров’я, то замість операції додавання – значення «health» буде прирівняно до «MAX_HEALTH». Візуальне відображення роботи даного методу, у вигляді схеми можна побачити на рисунку 1.22:

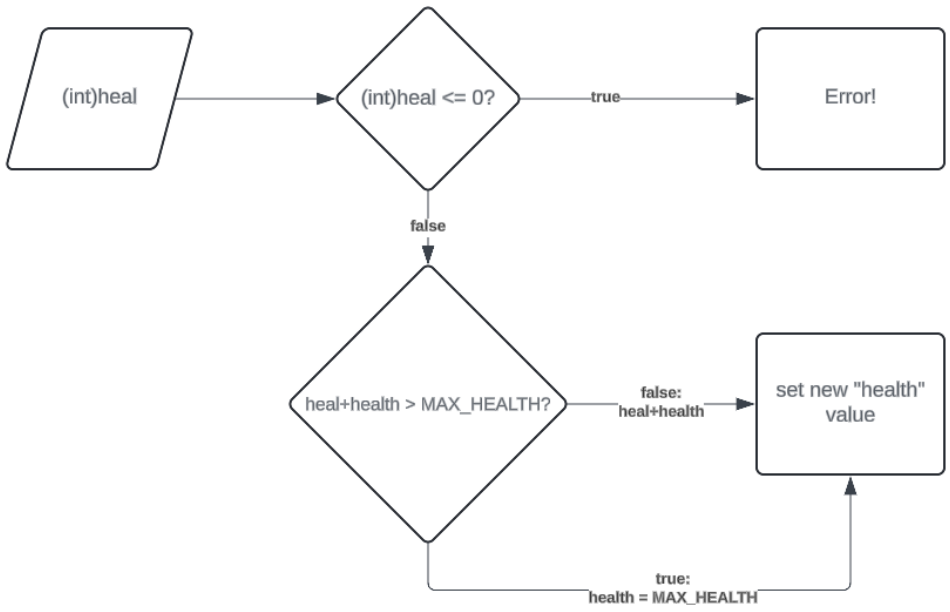


Рисунок 1.22. Схема роботи методу «Heal»

Наступним треба реалізувати метод «Death». Цей метод необхідний задля того щоб у випадку, коли рівень здоров’я носія скрипту падає до нулю або нижче – його було видалено зі сцени. Рядок «Destroy(gameObject)» при виклику даного методу видаляє ігровий об’єкт, до якого прикріплений даний скрипт, в той час як «Debug.Log("I'm dead x_x")» виводить повідомлення у консоль, яке свідчить про успішне виконання команди.

Тепер, коли скрипт має необхідний проекту функціонал – залишилось додати його до об’єкту «Player», та виставити необхідні параметри. Процес

додавання зображено на рисунку 1.23:

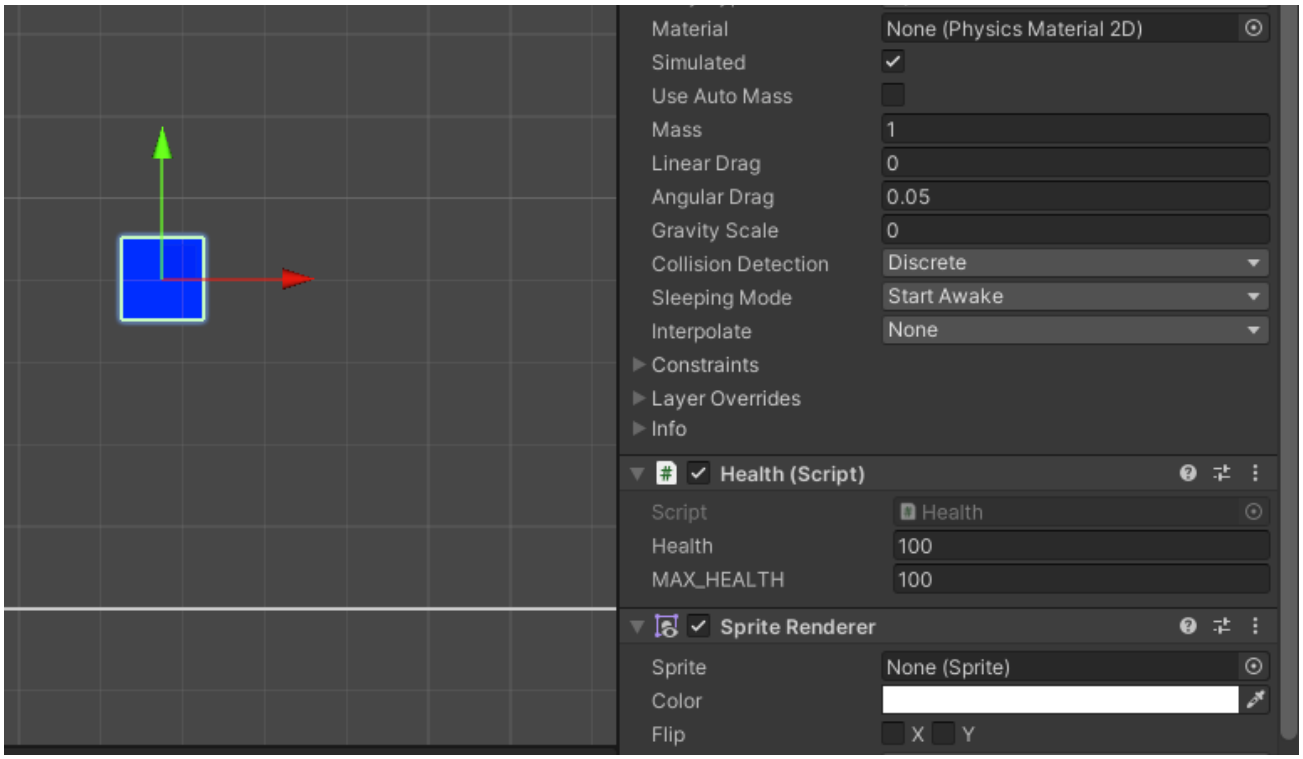


Рисунок 1.23. Додавання скрипту «Health» до об’єкту «Player»

1.5.3 Реалізація ворогів і їх штучного інтелекту

Тепер треба додати до проекту ворогів, які протистоятимуть гравцю. Для цього створюємо такий самий квадрат, як і для гравця на початку, через ієрархію об’єктів та змінюємо його колір на червоний, для подальшої відміни від гравця і додаємо квадратний коллайдер. Дублюємо його та робимо один менше, а інший – більше. Тепер, коли на сцені є два червоних квадрата – перейменовуємо їх в «enemy» та «BigEnemy» відповідно.

Тепер, лишаючи заготовлені пустушки на сцені – створюємо нову папку «AI», за директорією Assets/Scripts та в ній створюємо ще одну папку «Hostile». Ця папка зберігатиме скрипти для ботів із штучним інтелектом, що налаштовані на агресивну поведінку відносно гравця. Тепер створюємо в ній скрипт «Enemy» та «EnemyData».

Для початку слід відредагувати «EnemyData». Він буде відрізнятиметься від інших, тим що буде являтись скриптом типу «ScriptableObjects». Даний тип скриптів призначений для зберігання інформації, в даному випадку для задання

характеристик різним типам противників. Виглядає цей скрипт наступним чином:

```
[CreateAssetMenu(fileName = "Data", menuName = "ScriptableObjects/Enemy", order = 1)]
public class EnemyData : ScriptableObject
{
public int HP;
public int damage;
public float speed;
}
```

Після цього треба створити нову папку в «Assets» під назвою «Enemies». Ця папка зберігатиме заздалегідь збережену інформацію о різних типах ворогів у вигляді асетів, дозволяючи легко вносити зміни та додавати нові типи ворогів у майбутньому. Створення нових асетів, за допомогою даного скрипту, зображено на рисунку 1.24:

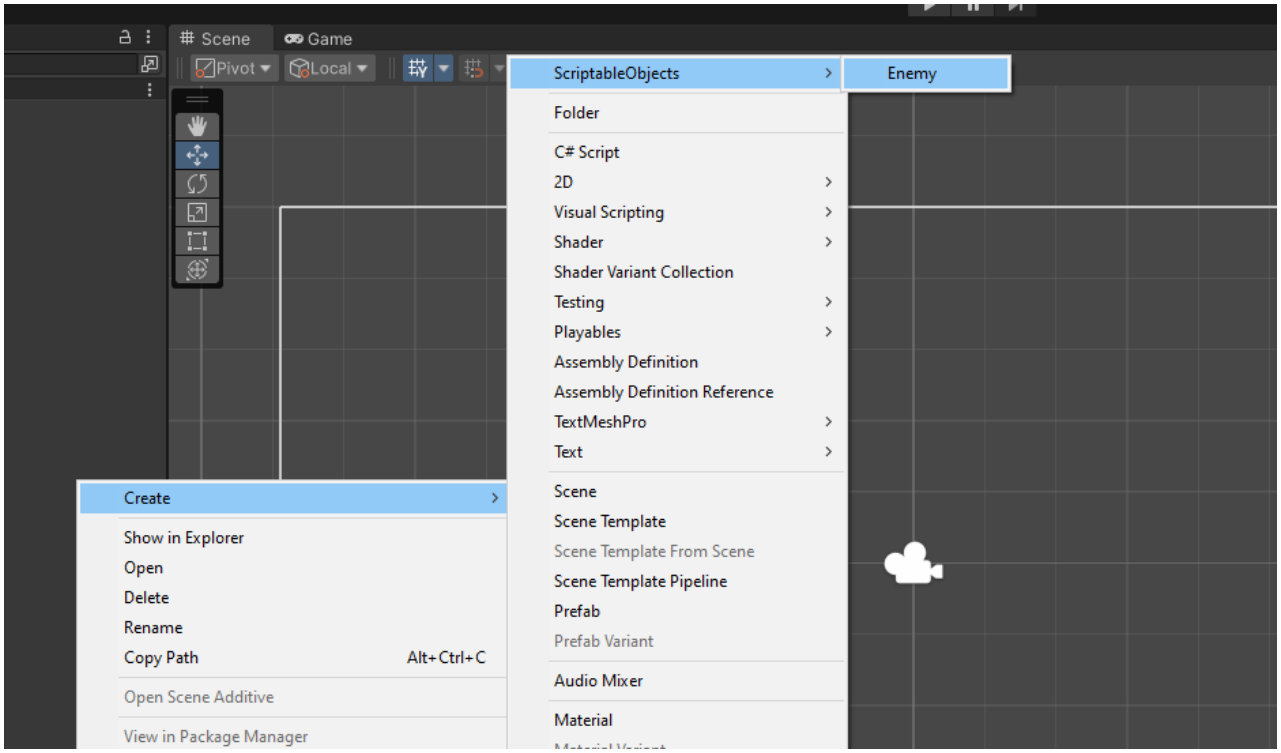


Рисунок 1.24. Створення нового асету з допомогою ScriptableObjects

Створюємо два асети: «Swarmer» та «BigSwarmer». Вносимо до них зміни:

- BigSwarmer: HP - 18; Damage - 10; Speed - 1;

- Swarmer: HP - 9; Damage - 5; Speed – 1.5;

Надалі дані асети можна буде використовувати для створення різних типів ворогів з різними параметрами. Створені асети зображено на рисунку 1.25:

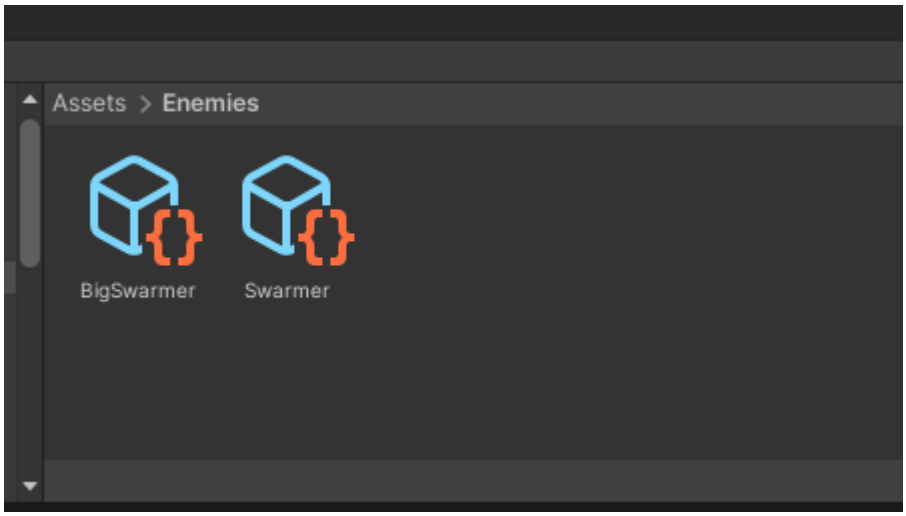


Рисунок 1.25. Інформаційні асети у папці «Enemies»

Після створення асетів, можна перейти до створення скрипту ворогів. Для цього відкриваємо раніше створений скрипт «Enemy» для подальшого внесення змін.

```
public int damage = 5;
public float speed = 1.5f;
public EnemyData data;
public GameObject player;
void Start()
{
player = GameObject.FindGameObjectWithTag("Player");
SetEnemyValues();
}
void Update()
{
Swarm();
}
```

Вище приведена перша половина скрипту. Скрипт починається з оголошення необхідних змін. Змінні «damage» та «speed», які відповідають за значення урону та швидкості, оголошені для визначення стандартного значення цих характеристик, у випадку якщо вони не будуть визначенні на початку роботи скрипту. Змінна «data» необхідна для визначення характеристик конкретного ворога, а «player» необхідна для пошуку гравця.

В початковому методі «Start», який спрацьовуватиме при ініціалізації скрипту, рядок «player = GameObject.FindGameObjectWithTag("Player");» шукає

ігровий об’єкт з тегом «Player», та записує його в змінну «Player», для подальшої взаємодії, а також викликається метод «SetEnemyValues», описаний нижче, як і метод «Swarm» викликаний в «Update».

```
private void SetEnemyValues() {
GetComponent<Health>().SetHealth(data.HP, data.HP);
damage = data.damage;
speed = data.speed;
}
private void Swarm() {
transform.position = Vector2.MoveTowards(transform.position,
player.transform.position, speed * Time.deltaTime);
}
private void OnTriggerEnter2D(Collider2D collider)
{
if (collider.CompareTag("Player")) {
if (collider.GetComponent<Health>() != null) {
collider.GetComponent<Health>().Damage(damage);
}
}
}
}
```

В другій половині коду прописані три методи, що становлять функціонал даного скрипту: «SetEnemyValues», «Swarm» та «OnTriggerEnter2D»

«Swarm» – використовуючи метод «Vector2.MoveTowards», заставлятиме ворога рухатись в сторону гравця зі швидкістю, визначеною змінною «speed».

Тим часом, SetEnemyValues бере інформацію записану в заздалегідь створені асети, в папці «Enemies» та встановлює відповідні параметри, залежно від типу ворога. Після додавання скрипту до об’єкту – необхідно перетягнути відповідний асет, задля його додання в відповідне поле і запису необхідних параметрів в змінні скрипту.

Метод OnTriggerEnter2D спрацьовує, якщо в трігерний коллайдер носія скрипту потрапляє інший коллайдер. Він перевірятиме наявність тегу «Player» і компоненту «Health» в об’єкту, з яким зіштовхується. При успішному проходженні обох перевірок – в компонента «Health» викликається метод «Damage», в який передається змінна «damage» носію скрипта.

На рисунку 1.26 зображено прикріплений до ігрового об’єкту «Enemy»

відповідний скрипт, з визначеним асетом «Enemy Data», а на рисунку 1.27 зображена схема роботи методу «OnTriggerEnter2D»:

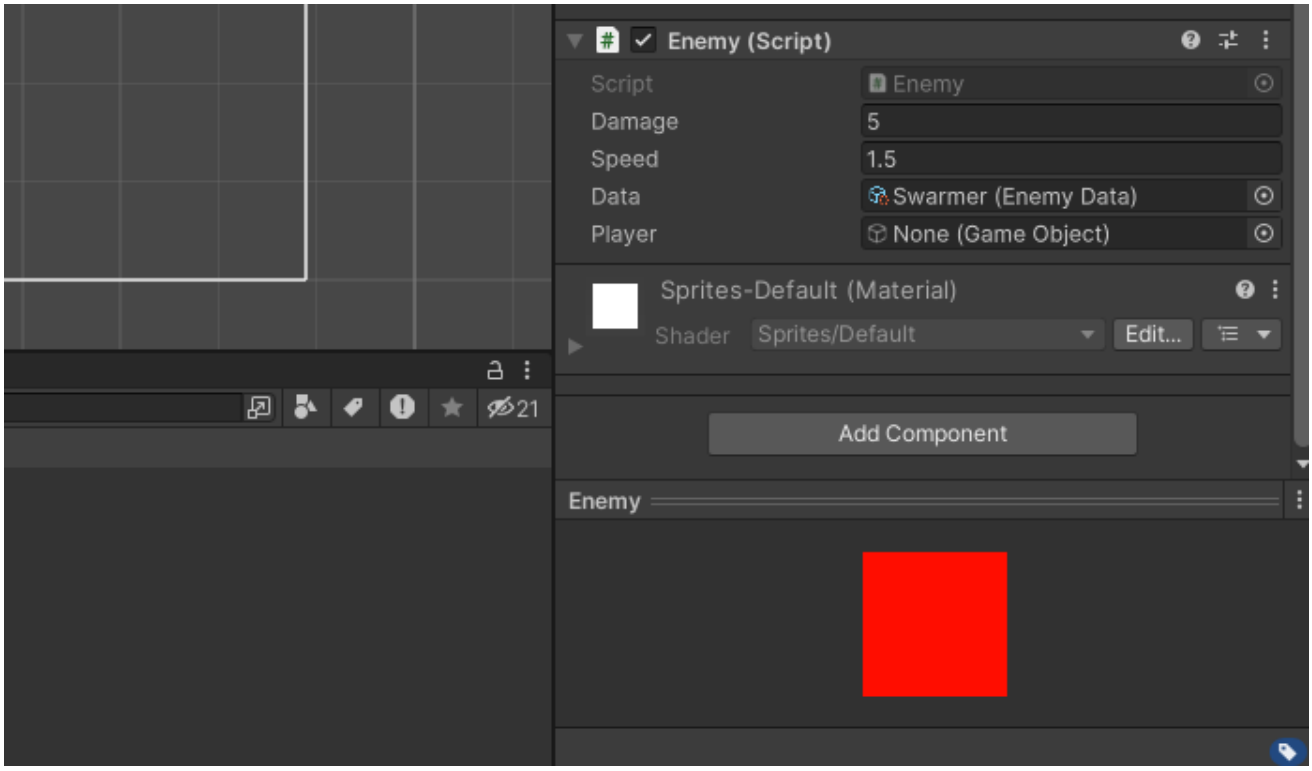


Рисунок 1.26. Скрипт «Enemy» з визначеним асетом в полі «Data»

Рисунок 1.27. Схема роботи методу «OnTriggerEnter2D»

Тепер, після додавання скрипту до раніше створених маленького і великого квадратів, вони слідуватимуть за гравцем, та наноситимуть йому урон при контакті.

Далі потрібно створити папку «Prefabs», в яку треба перетягнути обидва об’єкти, для створення шаблонів, або префабів цих об’єктів. Ці префаби необхідні задля більш легкого створення копій цих об’єктів.

1.5.4 Реалізація системи генерації ворогів

Тепер, коли в проекті присутні префаби обох типів ворогів – треба впровадити щось, що буде створювати їх копії на сцені. Дану функцію буде виконувати спеціальний ігровий об’єкт – спавнер.

Для початку треба створити новий примітив, який відображатиме спавнер або точку появи на сцені. Через контексте меню створюємо білий шестигранник, та перейменовуємо його в ієрархії об’єктів на «Spawner». Далі в директорії Assets/Scripts/AI/Hostile створюємо C# скрипт, з відповідною до імені об’єкту назвою.

Даний ігровий об’єкт має:

- Генерувати певну кількість очок «тікетів», які даний об’єкт потім витрачатиме на генерацію групи ворогів, які коштуватимуть різну кількість очок, залежно від їх типу;

- Витрачати ці очки, розподіляючи їх на великих та маленьких ворогів та записуючи те, яких ворогів об’єкт буде генерувати;

- Генерувати ворогів на місці розташування об’єкту та відповідно тому, як були витрачені тікети – випускати їх на сцену один за одним.

Спочатку скрипт має сгенерувати певну кількість тікетів. За це буде відповідати метод «Tickets», приведений нижче.

```
private int Tickets(int rounds) {
int result = rounds + 10;
Debug.Log("Сгенерував " + result + " тікетів");
return result;
}
```

Даний скрипт приймає на вхід числову змінну «rounds» і повертає числову змінну «result», яка обчислюється за формулою n+10, де «n» - це кількість «раундів», які протримався гравець.

Далі йде метод «EnemyTypesGenerator», який розподіляє отриману кількість тікетів на різні типи ворогів. Нижче приведено рисунок 1.28, який відображає схему роботи спавнеру і код «EnemyTypesGenerator» даного методу:

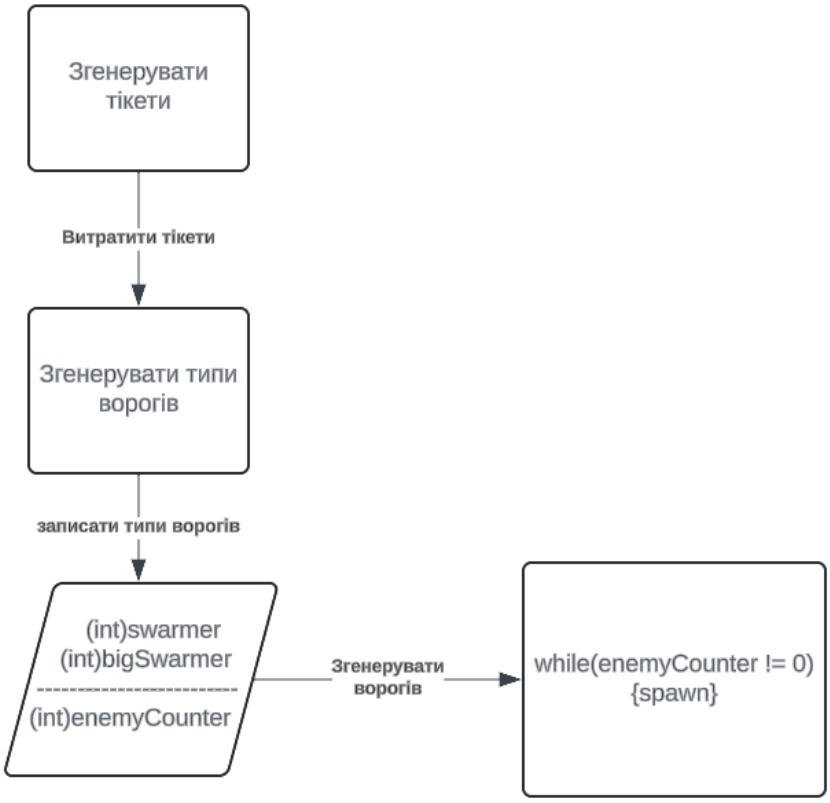


Рисунок 1.28. Схема роботи спавнеру ворогів

```
private void EnemyTypesGenerator(int tickets) {
while (tickets > 0) {
int random = Random.Range(1, 3);
if (random == 1)
{
swarmer++;
tickets--;
Debug.Log("маленького ворога згенеровано");
}
else
{
bigSwarmer++;
tickets--;
Debug.Log("великого ворога згенеровано");
}
}
}
```

Для довільної генерації ворогів даний метод використовує власну змінну «random», значення якій присвоює метод Random.Range(1, 3). Далі метод перевіяє

значення змінної, і якщо «random» дорівнює 1 – до лічильнику маленьких ворогів додається 1, а від кількості тікетів віднімається 1. У випадку, якщо «random» дорівню 2 – збільшується лічильник великих ворогів, а лічильник тікетів також зменшується.

Вищеперераховані дії відбуваються у циклі «while», який закінчується, коли кількість тікетів падатиме до нулю.

Потім спрацьовуватиме метод «Spawn», який відповідає за безпосередньо генерацію ворогів на сцені.

```
private void Spawn() {
if (swarmer + bigSwarmer == 0)
{
StopSpawning();
return;
}
int random = Random.Range(1, 3);
if (random == 1 || swarmer != 0)
{
//gen swarmer
Instantiate(swarmerPrefab, new Vector3(this.transform.position.x,
this.transform.position.y, this.transform.position.z), Quaternion.identity);
swarmer--;
enemyCounter++;
Debug.Log("маленького ворога додано на сцену");
}
else
{
//gen bigSwarmer
Instantiate(bigSwarmerPrefab, new Vector3(this.transform.position.x,
this.transform.position.y, this.transform.position.z), Quaternion.identity);
 bigSwarmer--;
enemyCounter++;
Debug.Log("великого ворога додано на сцену");
}
}
```

Даний метод містить в собі змінну «random», яка схожим на «EnemyTypesGenerator» способом генерує довільне значення(1 чи 2) і на його основі генерує на сцені ворога.

Також цей метод використовує інший метод – «StopSpawning», у випадку якщо всі необхідні вороги вже були створені на сцені. Окрім нього є ще метод «StartSpawning», який напроти – починає генерацію ворогів на сцені. Виглядають ці методи наступним чином:

```
private void StartSpawning()
{
spawning = true;
InvokeRepeating("Spawn", 2f, 1.5f);
Debug.Log("спавн почав");
}
private void StopSpawning()
{
spawning = false;
CancelInvoke("Spawn");
Debug.Log("спавн закінчив");
}
```

Принцип роботи цього відрізку коду полягає в методі «InvokeRepeating». Цей метод дозволяє викликати певний метод раз у визначений проміжок часу, що дозволяє випускати ворогів на сцену по одному.

1.5.5 Реалізація піксельної графіки

Для впровадження графіки у проект існує два можливі шляхи її реалізації:

- Пошук готових вихідних матеріалів в інтернеті, їх придбання, якщо вони платні, та завантаження і подальше впровадження їх в проект.

- Пошук та встановлення спеціалізованого програмного забезпечення, для створення графіки, та самостійне створення вихідних матеріалів для проекту, задля їх подальшого впровадження.

Для реалізації графіки в даному проекті був обраний другий варіант, зі створенням власних матеріалів.

Для створення власних графічних файлів, які в подальшому можна буде використовувати, спочатку потрібно обрати відповідне програмне забезпечення, яке допоможе в цій справі. Прекрасним варіантом для створення піксельної графіки є додаток «Aseprite».

Aseprite – це потужний інструмент для створення піксельної графіки та

анімації, який здобув популярність серед художників, розробників ігор та аніматорів завдяки своїм широким можливостям і зручному інтерфейсу. Aseprite надає все необхідне для створення складних графічних проектів, забезпечуючи користувачів інструментами для малювання, анімації та редагування. На рисунку 1.29 приведений приклад роботи над проектом в «Aseprite»:

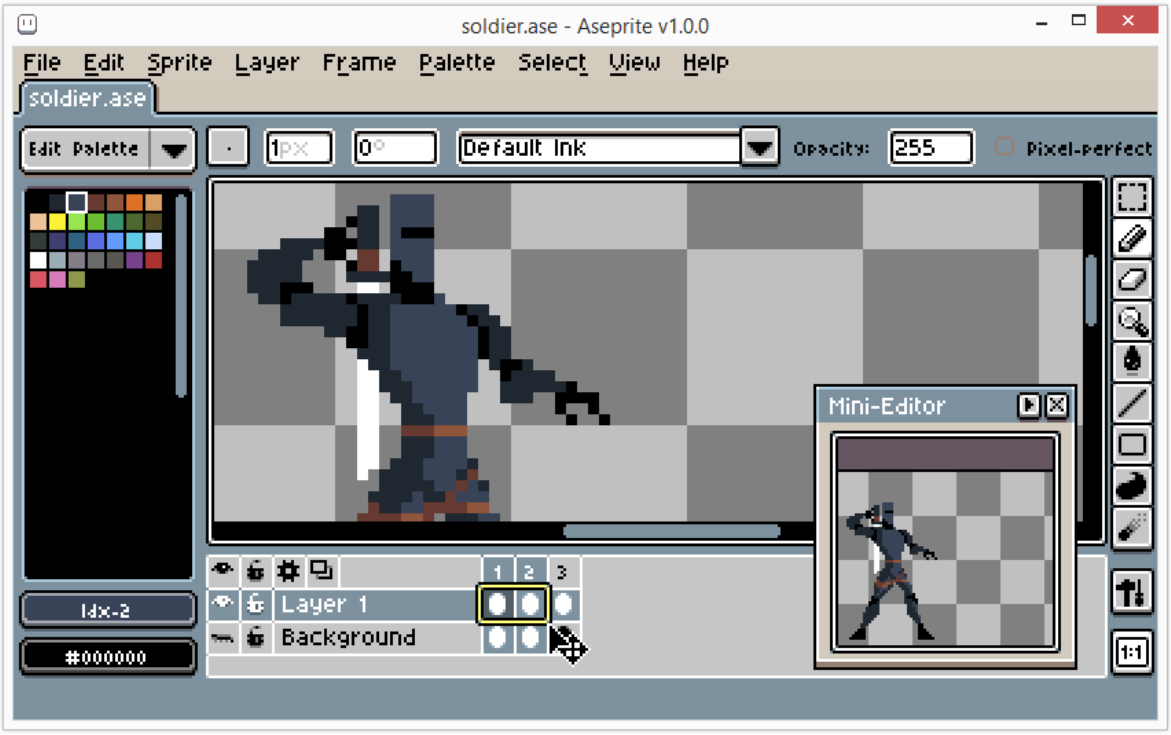


Рисунок 1.29. Приклад роботи в Aseprite

Він був створений програмістом і художником Давидом Капелло (David Capello). Цей проект розпочався у 2001 році під назвою "Allegro Sprite Editor", або просто ASE, як програмне забезпечення з відкритим вихідним кодом. Протягом років, Aseprite еволюціонував, розширюючи свої функції та можливості, і у 2014 році проект було перейменовано на Aseprite.

Aseprite є прекрасним вибором для створення піксельної графіки не тільки для професійних художників, а й, що досить важливо, для початківців і аматорів цієї справи, через загальну простоту і дружелюбність інтерфейсу і процесу.

Встановивши і відкривши Aseprite – необхідно створити новий файл, через File > New, після чого з’являється вікно налаштування нового проекту, зображене на рисунку 1.30, після взаємодії з яким створено полотно розміром 16x16 пікселів, зображене на рисунку 1.31:

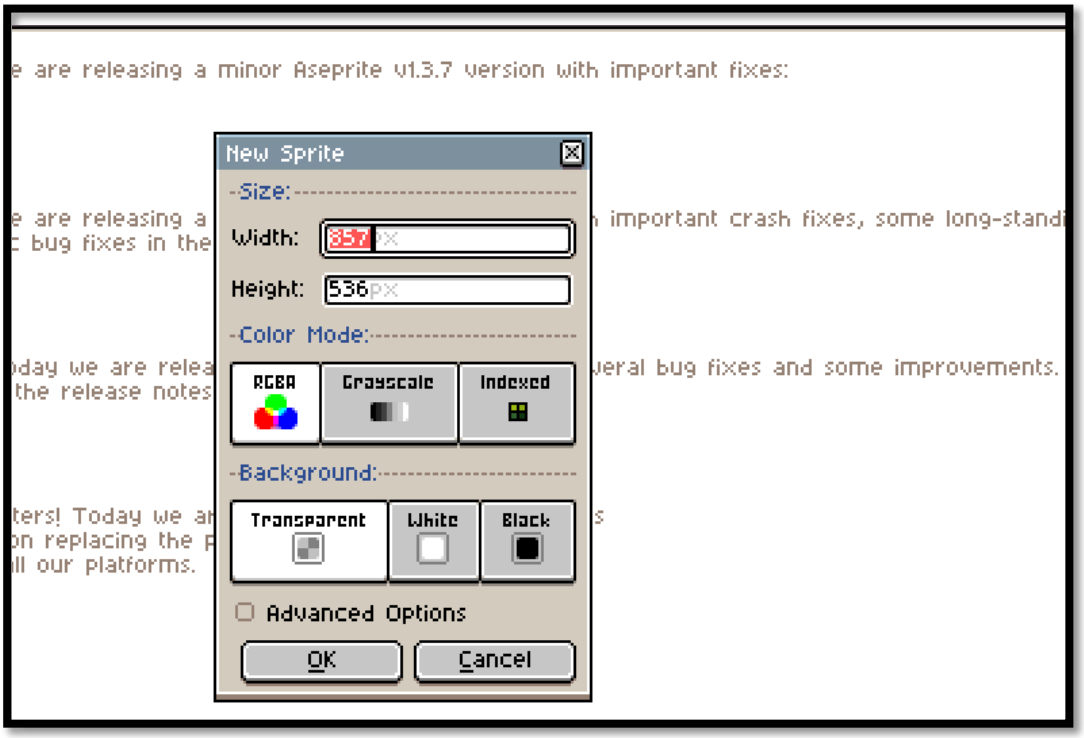


Рисунок 1.30. Налаштування нового проекту в Aseprite

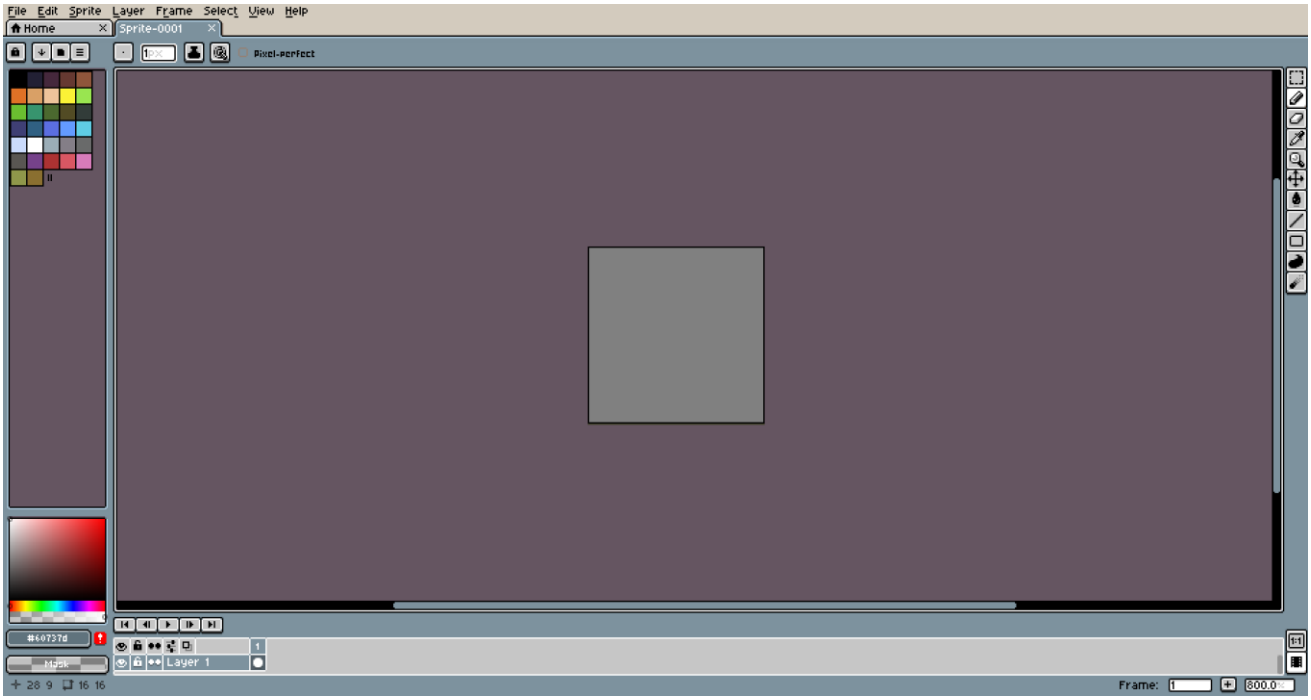


Рисунок 1.31. Пустий файл спрайту 16x16

Тепер треба визначити, що саме треба намалювати. Необхідні проекту спрайти можна розділити на дві групи – персонажі та фонові об’єкти. Персонажі – це головний герой, яким управлятиме гравець, та вороги, які протистоятимуть йому, в той час коли фонові об’єкти – це все, від фону, на якому ходять персонажі,

до декоративних об’єктів розташованих на сцені, таких як, наприклад, кущі, дерева, тощо.

Так як дії гри проходитимуть в лісу – потрібно намалювати відповідний візуал. Якщо для фону, який буде зображати траву, підходитиме монолітний зелений колір, то декоративні об’єкти повинні представляти з себе дерева, каміння, велике та маленьке і кущі. Також гравець, з’являючись в центрі карти, буде з’являтись з порталу, а вороги, з якими йому доведеться битись – з печери, тому ці об’єкти також треба намалювати.

Першим кроком буде створення спрайтів для оточення. За допомогою інструмента заливання – заливаємо весь простір спрайту зеленим кольором. Це спрайт можна буде розтягнути на всю сцену і використовувати як трав’янистий фон. Зберігаємо файл з назвою «grass», зображений на рисунку 1.32:

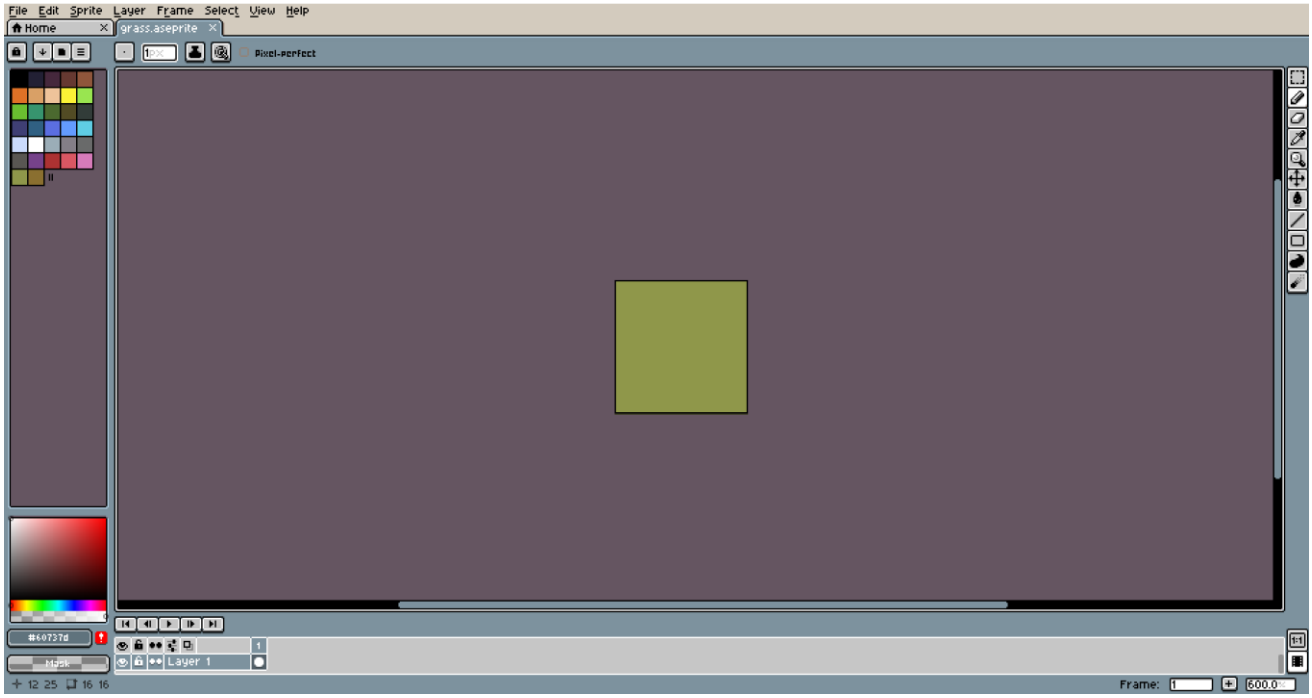


Рисунок 1.32. Створення спрайту «Grass»

Далі малюємо спрайт для дерева. Створивши новий спрайт розміром 16x16 – вимальовуємо сам спрайт. Також для створення ярко вираженого контуру використовуємо чорний колір. Результат зображено на малюнку 1.33:

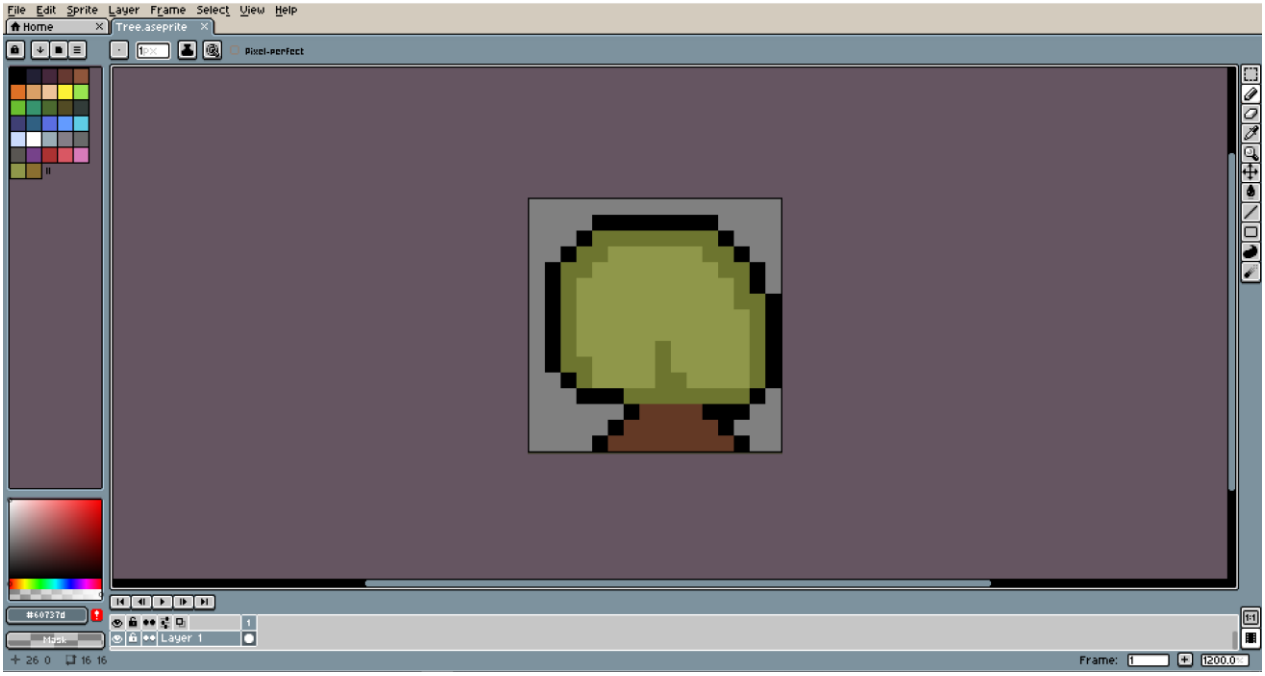


Рисунок 1.33. Створення спрайту «Tree»

Наступним кроком буде створення спрайтів для великих та маленьких каменів. Вони будуть використовувати подібну кольорову палітру, що в декілька разів спрощує роботу над ними. Результат роботи на рисунках 1.34 і 1.35:

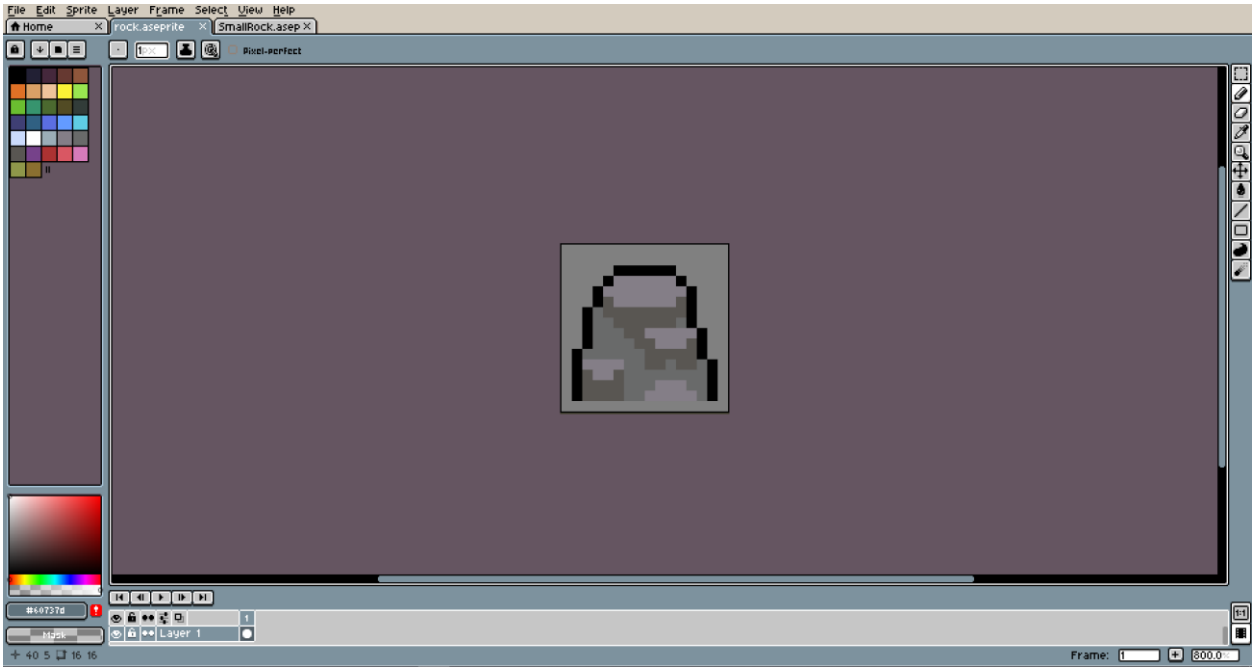


Рисунок 1.34. Створення спрайту «Rock»

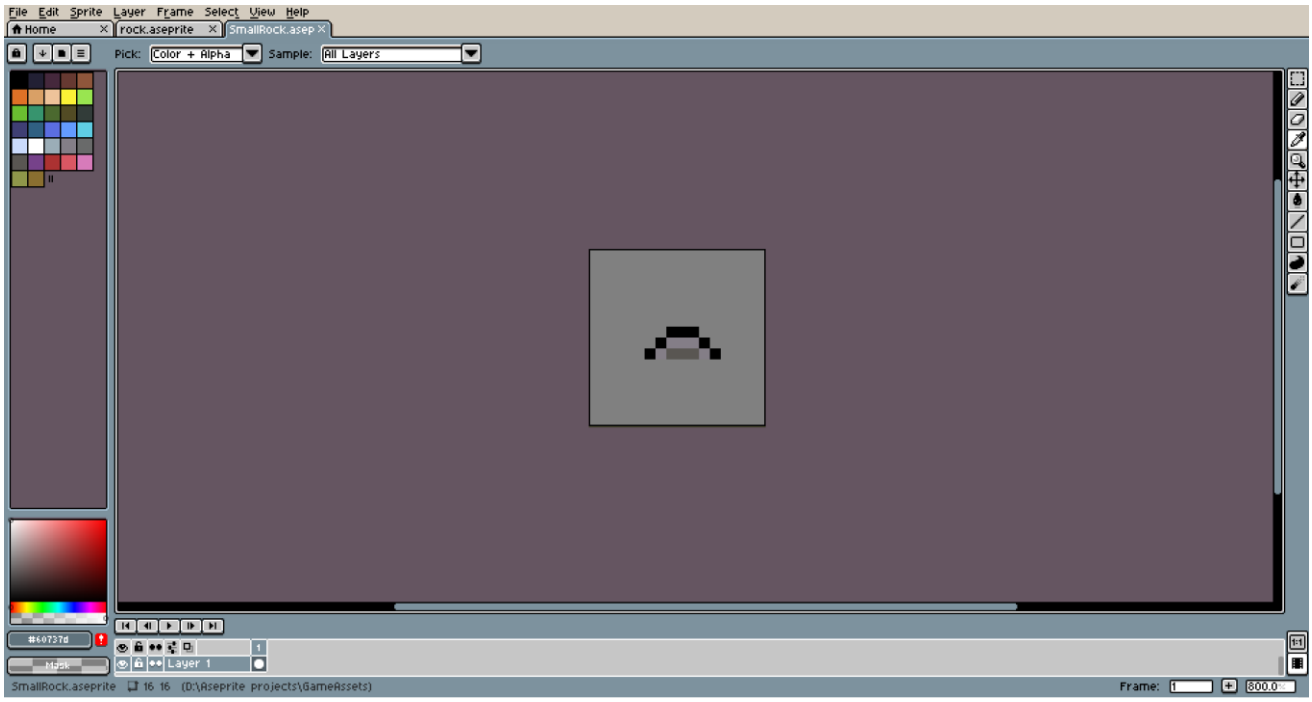


Рисунок 1.35. Створення спрайту «SmallRock»

Наступним об’єктом для малювання буде кущ. Для того, щоб він не вибивався із палітри, цей спрайт використовуватиме ті ж кольори що і «Tree». Намальований спрайт куща зображено на рисунку 1.36:

Рисунок 1.36. Створення спрайту «Bush»

Далі треба намалювати спрайти персонажів. На рисунку 1.37 приведений результат роботи над спрайтом для герою:

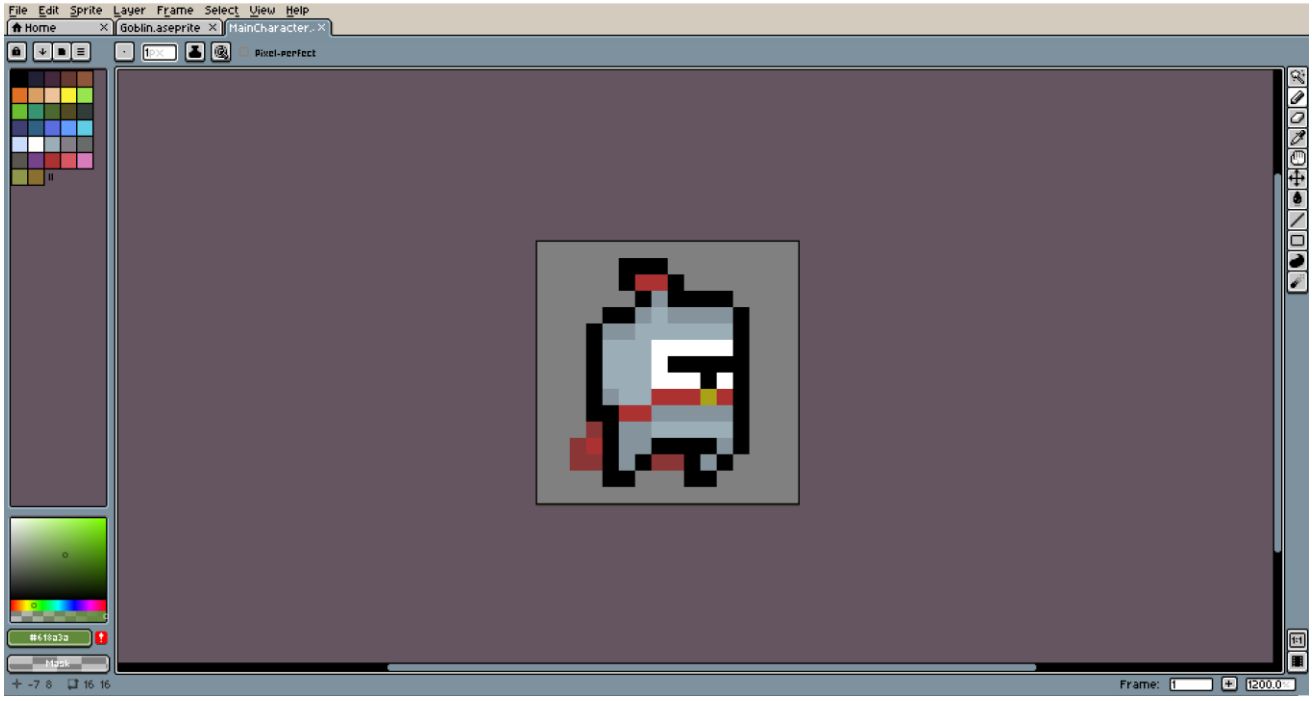


Рисунок 1.37. Створення спрайту «MainCharacter»

Даний спрайт зображує головного героя, який був натхнений образом типового лицаря середньовіччя, у повних латах. Тим часом, результат роботи над спрайтом гобліну зображено на рисунку 1.38:

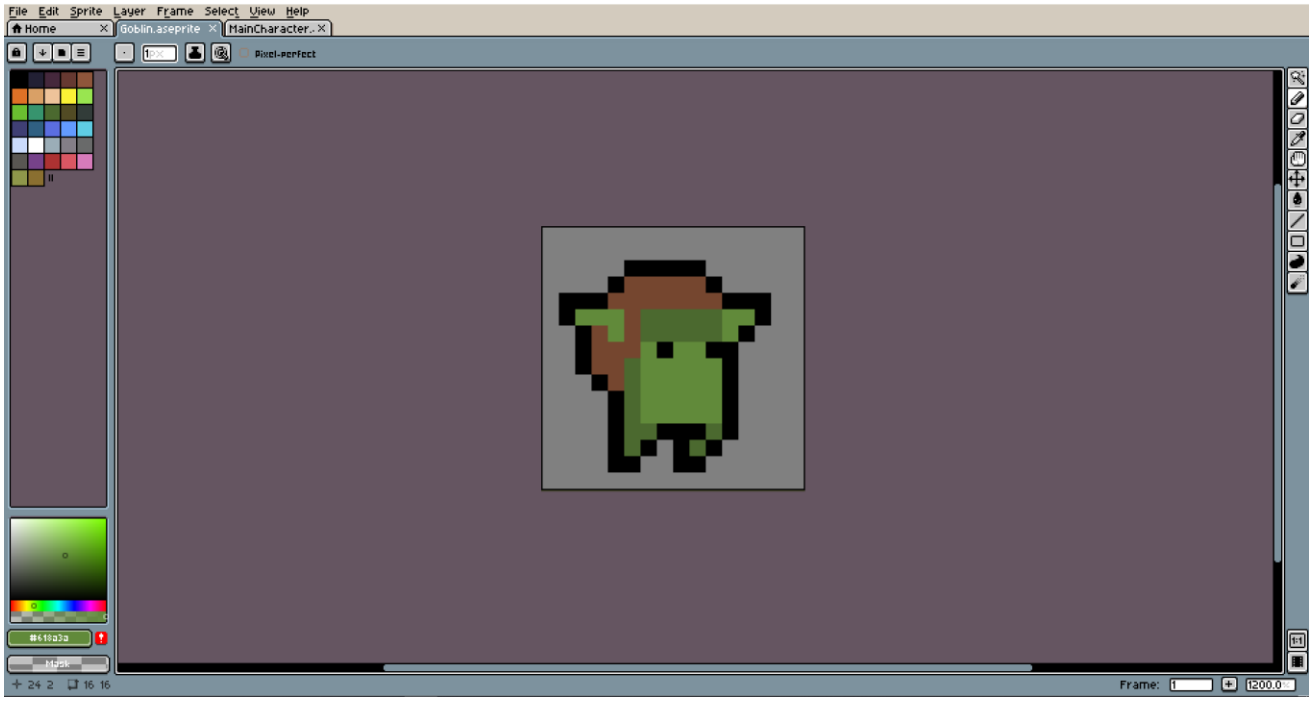


Рисунок 1.38. Створення спрайту «Goblin»

Після створення спрайтів для обох персонажів – наступними потрібно намалювати точки їхньої появи на ігровій сцені. Для головного героя – це портал,

зображений на рисунку 1.39, а для ворогу-гобліну – це печера, яка зображена на рисунку 1.40:

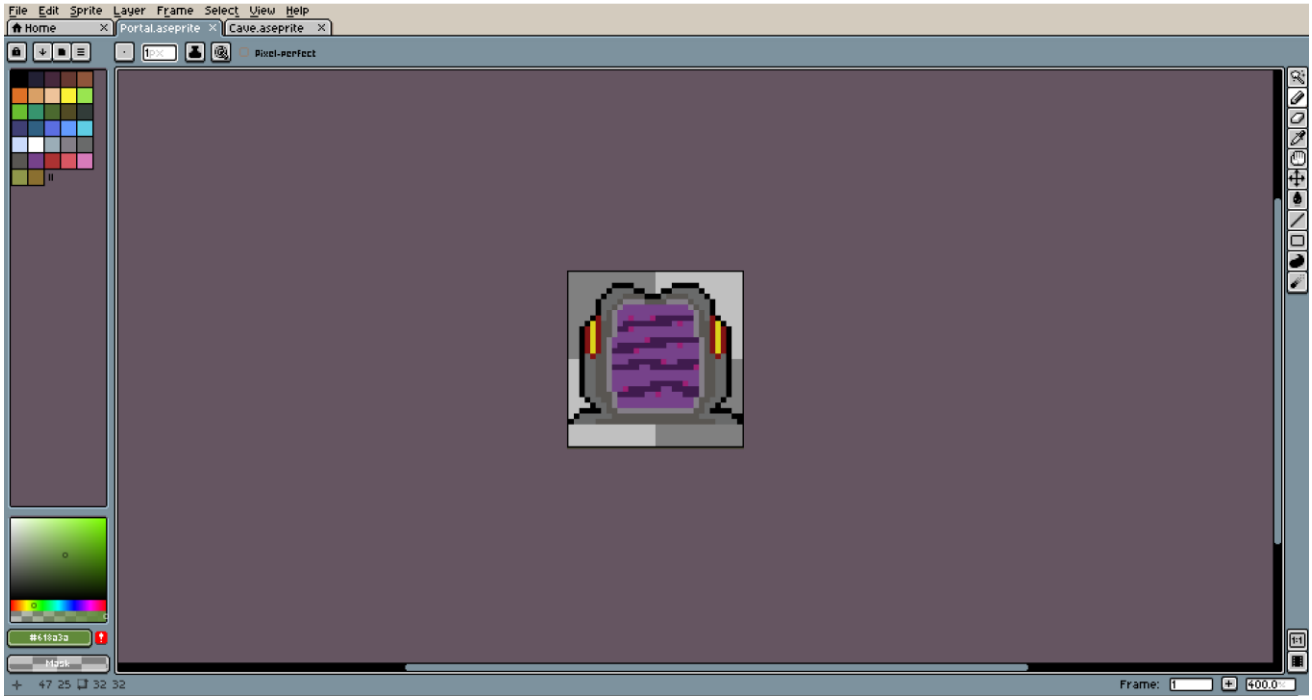


Рисунок 1.39. Створення спрайту «Portal»

Спрайт порталу, як і спрайт печери, головним чином відрізняються від усіх попередніх спрайтів тим, що вони малюються на полотні розміром 64x64. Ця різниця обумовлена тим що ці об’єкти повинні бути помітно більші за попередні.

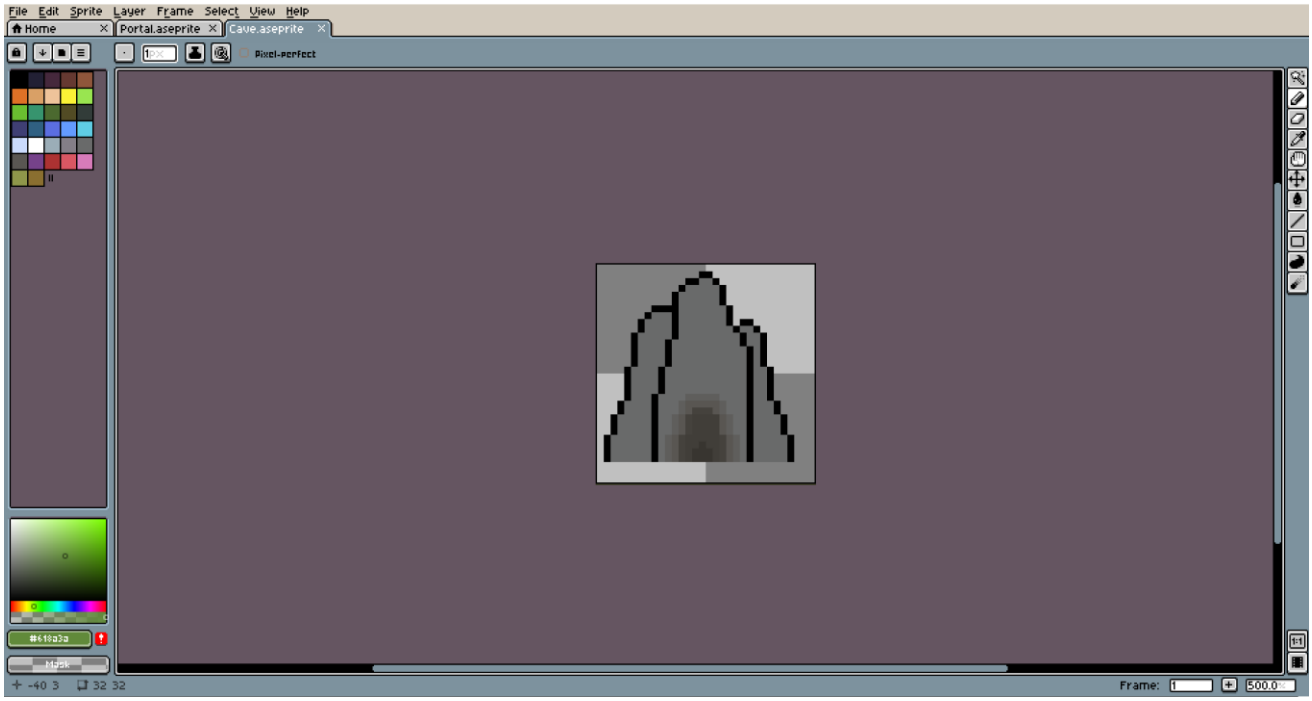


Рисунок 1.40. Стовреня спрайту «Cave»

Тепер, коли усі необхідні спрайти були намальовані – треба повернутись назад до проекту, куди їх час імпортувати. Для цього треба створити папку «Sprites» та перенести до неї всі спрайти намальовані раніше спрайти, для подальшої роботи з ними.

Прикріпляємо і дивимось на роботу проекту з застосованими до об’єктів спрайтами. На рисунку 1.41 зображений робочий проект з прикріпленими спрайтами:

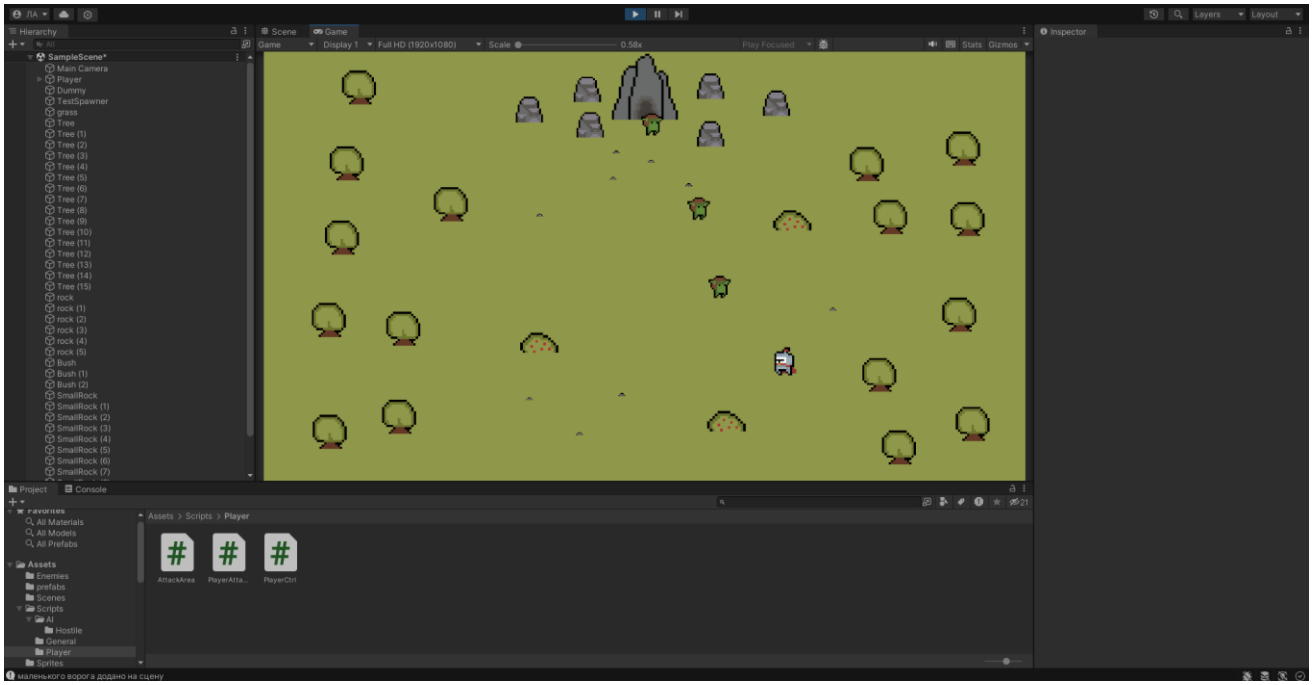


Рисунок 1.41. Перевірка роботи проекту з реалізованою графікою

1.6 Тестування працездатності ігрових елементів

Тестування працездатності продукту – це критично важливий етап розробки. Він гарантує, що всі компоненти гри функціонують належним чином, відповідають заданим вимогам і надають користувачам якісний ігровий досвід. Процес тестування включає ретельний аналіз всіх елементів проекту і перевірку працездатності його функцій у різноманітних сценаріях і випадках, які можуть надати умови гри.

Обширні тестування проекту протягом розробки виявляли певні проблеми. Першою з них було те, що модель гравця, яка в першій половині процесу розробки являла собою примітив-куб, розтягувалась по горизонталі, при русі по

горизонталі. Причиною даної проблеми було те, що в методі «Check4Flipping», коли гра виконує скрипт «PlayerCtrl» – вона, в той самий момент, як гравець починає рух по горизонталі, застосовує зміни до ігрового об’єкту «Player», а саме за допомогою команди «transform.localScale». При русі вліво масштаб об’єкту за координатою «X» змінюється на «-1», а при русі вправо напроти – на «1», що не збігається з масштабом об’єкту «0.75», що і призводить до «розтягування» ігрового об’єкту. Для вирішення даного багу необхідно або змінити розміри об’єкту на сцені під значення, записане в коді, або модифікувати код, під характеристики об’єкту. Перший варіант підходить при роботі з проектом на ранніх його етапах через легкість і швидкість внесення змін за допомогою редактору Unity, проте при додаванні повноцінної графіки до проекту – другий варіант є більш оптимальним, оскільки необхідно дотримуватись візуального стилю і рамок, які він накладає.

Ще одною проблемою, яка була виявлена під час роботи, було те, що спавнери генерували ворогів генерували їх по координатам, вказаних в префабах ворогів, замість координат самого спавнеру. Причиною даної проблеми було те, що в методі «Instantiate», не було вказано координат для генерації копій вказаних префабів, через що гра використовувала для цього збережені в префабах координати. Це вирішилось додаванням посилань на координати носія скрипту(спавнеру), за допомогою «this.transform.position». Окрім цього, спавнери також безкінечно генерувати ворогів. Це було вирішено впровадженням методів StartSpawning» та «StopSpawning» які ізолюють і спрощують контролювання початку і завершення генерації ворогів; виправлення умовних операторів «if», які перевіряють вимоги для початку/завершення генерації.

Подальше тестування проекту на кінцевих етапах розробки, не виявило ніяких критичних помилок в його роботі.

2 ЕКОНОМІЧНИЙ РОЗДІЛ

2.1 Резюме

В даному дипломному проекті була розроблена та реалізована 2D-гра у жанрі «слешер» на програмному рушії Unity. В процесі розробки було проведено глибокий аналіз вже існуючих ігрових рішень на ринку в даному жанрі, та оцінено можливість впровадження вдалих елементів. Були реалізовані базові(«core») механіки, необхідні для реалізації ігрового процесу згідно концептуального бачення проекту. Використання принципів об’єктно орієнтованого програмування при розробці дозволило досягти певного рівня гнучкості архітектури проекту, для облегшення впровадження потенційних модифікацій до нього.

Оцінка якості програмного продукту з точки зору користувача визначається необхідним на стадії функціонування розміром оперативної пам'яті ЕОТ, витратами машинного часу, пропускною спроможністю каналів передачі даних. Оцінка якості програмного продукту включає визначення трудомісткості і вартості його створення.

2.2 Визначення трудомісткості розробки програмного забезпечення

Тривалість розробки програмного продукту залежить від його обсягу, трудомісткості розробки, кваліфікації виконавців, а також планових термінів, визначених умовами ринку. Методом структурної аналогії по відповідних каталогах аналогів програмного забезпечення визначається обсяг програмних засобів, у тисячах умовних машинних команд програми аналога

Каталог аналогів

У таблиці 2.1 представлені аналоги програмного забезпечення, функції яких, у більшому або меншому ступені, виконує розроблений програмний продукт. Для нашого варіанта виділено сірим кольором.

Таблиця 2.1. Аналоги програмного забезпечення

Найменування ПЗ	*Обсяг функції ПП = V_0, ум. машинних команд*
1. ПП СУБД	1300 – 8600
2. ПП введення інформації	1800 – 8800
3. ПП оптимізаційних розрахунків	13000 – 10200

Вибравши аналог ПП, що містить Vo в умовних машинних командах, трудомісткості визначати на основі табл.2.2

Таблиця.2.2. Трудомісткість

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

На підставі отриманого значення, по довіднику, визначається укрупнена норма часу на розробку аналога програмного забезпечення (коректується поправочним коефіцієнтом враховуючої умови розробки ПП, тобто в умовах комп'ютера, Кк=0,7÷0,8): Тар = 244 х 0,8 = 195.20 (люд/годин).

Трудомісткість програмного продукту визначається по кожному етапу розробки окремо на підставі трудомісткості аналога з урахуванням складності розробки, ступеня новизни і ступеня використання в розробці стандартних модулів на підставі формул:

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

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

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

Для розрахунку необхідні наступні коефіцієнти:

Li – питома вага і-го етапу розробки (див. табл. 2.2.);

Кн – поправочний коефіцієнт, що враховує ступінь новизни (див. табл. 2.3.); Кт – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм (див. табл. 2.4.).

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

Для нашого варіанта виділено сірим кольором.

Таблиця 2.4. Значення поправочного коефіцієнта, що враховує ступінь новизни

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

Для нашого варіанта виділено сірим кольором.

Таблиця 2.5. Значення коефіцієнта ступеня використання в розробці типових програм

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

Для нашого варіанта виділено сірим кольором. Тепер розраховуємо трудомісткість по кожному етапу окремо:

Трудомісткість технічного завдання

$$Т_{тз}=Та*L_1*К_н=195,2 *0,12*0,7= 16,40 \text{ (люд/годин)}$$

Трудомісткість розробки технічного проекту

$$Т_{тп}=Та*L_2*К_н =195,2 *0,11*0,7 = 15,04 \text{ (люд/годин)}$$

Трудомісткість розробки робочого проекту

$$Т_{рп}=Та*L_3*К_н*К_т=195,2 *0,61*0,7*0,7= 58,35 \text{ (люд/годин)}$$

Для подальших розрахунків визначили кількість папера, витраченого на кожен етап: технічне завдання $N_{тз}$=2 (стр), розробка ТП $N_{тп}$=15(стр), розробка

робочого проекту $N_{рп}$=25 (стр), пояснювальна записка відповідно $N_{пз}$=50 (стр)

Розрахунок зведений у таблицю 2.6

Таблиця 2.6. Розрахунок трудомісткості ПП

Найменування етапів	Розрахунок, годин.		
1	2	3	4
1.ТЗ	$T_{Ртз}$=16,40	$T_{кк}=0,7*N_{тз}$= 0,7*2=1,4	$T_{нк}=0,15*N_{тз}$=0,15*2=0,30
2.Розробка ТП	$T_{Ртп}$=15,04	$T_{кк}=0,7*N_{тп}$=0,7*15=10,50	$T_{нк}=0,15*N_{тп}$=0,15*15=2,25
3.Розробка РП	$T_{Ррп}$=58,35	$T_{кк}=0,7*N_{рп}$=0,7*25=17,5	$T_{нк}=0,15*N_{рп}$=0,15*25=3,75
4.Розробка ПЗ	$T_{пз}=1,5**N_{пз}$=1,5*50 =75	$T_{кк}=0,7*N_{тз}$=0,7*50=35	$T_{нк}=0,15*N_{пз}$=0,15*50 =7.5
Усього, в т.ч.:	230,2		
- на розробку	$\Sigma T_р$=152		
- контроль керівника		$\Sigma T_{кк}$= 64,4	
- нормоконтроль			$\Sigma T_{нк}$=13,8

2.3 Розрахунок ціни програмного продукту

У цьому розділі для визначення ціни розраховуємо основну заробітну плату виконавців, матеріальні витрати, вартість машино – години і витрати на розробку ПЗ. Розрахунок основної заробітної плати виконавців приведений у таблиці 2.7. Відповідно до статті 8 «Закону про Державний бюджет України на 2024» встановлено мінімальну заробітну плату.

Таблиця 2.7 Розрахунок основної заробітної плати виконавців

Найменування робіт	Трудомісткість робіт, години	Погодинна тарифна ставка, грн.	Розрахунок, грн.
1.Розробка ПП	152	46,00	7084.00
2.Контроль керівника	65	46,00	2990,00
3.Нормоконт-роль	14	46,00	644,00
Усього	-	-	ΣЗо=10718,00

Зробимо розрахунок матеріальних витрат на розробку ПП. Розрахунок зведемо в таблицю 2.8.

Таблиця 2.8. Розрахунок матеріальних витрат на розробку ПЗ

Найменування матеріальних витрат	Тип, модель	Кількість	Ціна одиниці, грн.	Вартість, грн.
Папір	Лист А4	80	3,50	280,00
Разом	-	-	-	$В_{м\,і}$ =280
Транспортно– заготівельні витрати (10%)				$Втр\_з = 0{,}1 \times Вм1$= 0,1х280=28
Усього			308	$В_м=В_{мі}+В_{тр\_з=}$ 308

На підставі отриманих даних по окремих статтях витрат складена калькуляція планової собівартості в цілому ПП за формою, приведеною в таблиці 2.9.

Таблиця 2.9. Розрахунок статей витрат планової собівартості

Стаття витрат	Значення, грн.	Формула розрахунку
1. Матеріали	176,0	$В_м$ (див. табл. 2.7)
2. Основна заробітна плата	10718,00	$З_о$ (див. табл. 2.6)
3.Додаткова заробітна плата	1607,70	$Зд = 0{,}15 \times Зо$=10718х0,15
4.Відрахування до єдиного фонду соціального внеску	2711.65	$Вє.с.в. = 0{,}22 \times (Зо + Зд)$=0,22х (10718,00+ 1607,70)
5. Накладні витрати	6430.80	$Внак.$=0.6×$Зо$ = 0,6 х 10718,00
6. Повна собівартість	21644.15	$С_{пов}$= $В_м$ +$З_о$ + $З_д$ +Вє.с.в. + Внак.= 176,0+10718,00+1607,70+2711.65+ 6430.80

Розмір прибутку, що включається в ціну, визначаємо по наступній формулі:

$$П=(С_п*Р)/100= (21644.15*10)/100 = 2164.41грн \quad (2.4)$$

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

Оптова ціна (кошторисна вартість) визначається по формулі:

$$Ц_о = С_п+П = 21644.15+2164.41= 23808.56 \text{ грн} \quad (2.5)$$

Податок на додану вартість визначаємо по наступній формулі:

$$ПДВ = 0.2 * Ц_о = 23808.56 *0{,}2 = 4761.71грн; \quad (2.6)$$

Виходячи з отриманих даних, ціна реалізації розробленого програмного продукту на основі наступної формули, становитиме:

$$Ц_р=Ц_о+ПДВ = 23808.56 + 4761.71 = 28570.27 \text{ грн} \quad (2.7)$$

3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

3.1 Вступ

Охорона праці, як соціальний чинник, відіграє на підприємстві важливу, роль оскільки, якими б важливими не були трудові здобутки, вони не можуть компенсувати людині втраченого здоров'я, а тим більше життя. Те і інше дається лише один раз. Необхідно пам'ятати, що внаслідок нещасних випадків та аварій гинуть на виробництві не просто робітники та службовці, на підготовку яких держава вкладає значні кошти, а перш за все люди – годувальники сімей, батьки та матері дітей. Незадовільний стан охорони праці відображається на економіці держави. Служба охорони праці створюється на підприємствах незалежно від форми власності та видів діяльності для виконання правових, організаційно-технічних, санітарно-гігієнічних, соціально-економічних і лікувально-профілактичних заходів, спрямованих на запобігання нещасних випадків, професійних захворювань і аваріям в процесі праці. Основна мета всіх цих заходів – створити на підприємстві безпечні та здорові умови праці.

У розділі охорона праці дипломного проекту наведені характеристики приміщень, де експлуатуються ВДТ. До розгляду взято робоче місце програміста(оператора ЕОМ).

3.2 Аналіз небезпечних та шкідливих чинників, що впливають на працівника

Оператори ПК і програмісти зіштовхуються із впливом таких фізично небезпечних і шкідливих виробничих факторів, як підвищений рівень шуму, підвищена температура зовнішнього середовища, недостатня освітленість робочої зони, електричний струм та інші. Тому на робочому місці програміста повинні бути створені умови для високопродуктивної праці.

Перетворення і обробка інформації проводиться за допомогою ПК. Робота може кваліфікуватися як робота оператором ЕОМ.

3. 3 Розробка заходів з охорони праці

3.3.1 Виробничі приміщення

При плануванні виробничого приміщення врахована санітарна характеристика виробничих процесів, дотримуються норми корисної площі для працюючих, а також нормативи площ для розташування устаткування, що забезпечують безпечну роботу та зручне обслуговування устаткування.

Об'ємно-планувальні рішення будівель та приміщень для роботи з ВДТ мають відповідати вимогам ДСанПіН 3.3.2.007-98. Розміщення робочих місць З ВДТ ЕОМ і ПЕОМ у підвальних приміщеннях, на цокольних поверхах заборонено. Площа на одне робоче місце становить не менше ніж 6,0 $м^2$, а об'єм – не менше ніж 20,0$м^3$.

Виробничі приміщення повинні обладнуватися шафами для зберігання документів, полицями, стелажами, тумбами тощо, з урахуванням вимог до плащі приміщення.

У приміщеннях з ВДТ слід щоденно робити вологе прибирання. Приміщення повинні бути оснащені аптечками першої медичної допомоги.

3.3.2 Мікроклімат робочої зони працівників, вентиляція

У виробничих приміщеннях на робочих місцях з ВДТ мають забезпечуватись оптимальні значення параметрів мікроклімату: температури, відносної вологості й рухливості повітря (ДСанПіН 3.3.2.007-98).

Таблиця 3.1. Норми мікроклімату для приміщень з ВДТ ЕОМ та ПЕМ

Пора року	Категорія робіт	Температура повітря, С, не більше	Відносна вологість повітря %	Швидкість руху повітря, м/с
Холодна	Легка-1а	22-24	40-60	0,1
	Легка-1б	21-23	40-60	0,1
Тепла	Легка-1а	23-25	40-60	0,1
	Легка-1б	22-24	40-60	0,1

Рівні позитивних і негативних іонів у повітрі приміщень з ВДТ мають відповідати санітарно-гігієнічним нормам № 2152-80.

Таблиця 3.2. Рівні позитивних і негативних іонів

Рівні	Число іонів в 1 $см^3$ повітря	Число іонів в 1 $см^3$ повітря
	n+	n-
Мінімально необхідні	400	600
Оптимальні	1500-3000	3000-5000
Максимально допустимі	50000	50000

3.3.3 Освітлення робочого місця, шум, вібрація

Штучне освітлення в приміщеннях з робочими місцями, обладнаними ВДТ має здійснюватись системою загального рівномірного освітлення. У виробничих та адміністративних приміщеннях, у разі переважної роботи з документами, допускається застосування системи комбінованого освітлення – крім системи загального освітлення додатково встановлюються світильники місцевого освітлення.

Значення освітленості на поверхні робочого столу в зоні розміщення документів має становити 300-500лк.

Як джерела світла для штучного освітлення мають застосовуватись переважно люмінесцентні лампи типу ЛД. Допускається застосування ламп розжарювання у світильниках місцевого освітлення.

3.3.4 Організація робочого місця користувача ПК

Робочі місця слід так розташовувати відносно світових прорізів, щоб природне світло падало збоку, переважно зліва. При розміщенні робочих столів з ВДТ слід дотримуватися таких відстаней: між бічними поверхнями ВДТ -1,2м; від тильної поверхні одного ВДТ до екрану іншого – 2,5м.

Екран ВДТ має розташовуватися на оптимальній відстані від очей користувача, що становить 600…700 мм, але не ближче ніж за 600 мм з урахуванням розміру літерно-цифрових знаків і символів.

Клавіатуру розташовують на поверхні столу на відстані 100…300 мм від краю, зверненого до працюючого. У конструкції клавіатури має передбачатися опорний пристрій, який дає змогу змінювати кут нахилу поверхні клавіатури у межах $5…15^0$.

При оснащенні робочого місця лазерним принтером параметри лазерного випромінювання повинні відповідати вимогам СанПіН № 5804-91.

ЕОМ ВДТ і ПК , інше устаткування , електропроводи та кабелі за виконанням і ступенем захисту мають відповідати класу зони за НПАОП 40.1-1.01-97, мати апаратуру захисту від струму короткого замикання та інших аварійних режимів. У приміщеннях, де одночасно експлуатується понад п’ять ЕОМ встановлюється аварійний резервний вимикач, який може повністю вимкнути електричне живлення приміщення, крім освітлення. Не допускається підключати ЕОМ з ВДТ і ПК до звичайної двопровідної електромережі, в тому числі – з використанням перехідних пристроїв

3.3.5 Електробезпека

Це система організаційних і технічних заходів та засобів, що забезпечують захист людей від шкідливої і небезпечної дії електричного струму, електричної дуги, електричного поля і статичної електрики.

Основні технічні засоби і заходи забезпечення електробезпеки при нормальному режимі роботи електроустановок включають:

- ізоляцію струмовідних частин;
- недоступність струмовідних частин;
- блоківки безпеки;
- засоби орієнтації в електроустановках;
- виконання електроустановок, ізольованих від землі;
- захисне розділення електричних мереж;
- компенсацію ємнісних струмів замикання на землю;
- вирівнювання потенціалів.

Із метою підвищення рівня безпеки, залежно від призначення, умов експлуатації і конструкції, в електроустановках застосовується одночасно більшість з перерахованих технічних засобів і заходів.

Особа відповідальна за електрогосподарство призначається з числа працівників, які мають не нижче IV групи з електробезпеки та відповідний стаж роботи для обслуговування електроустановок несе персональну відповідальність за допущення працівника використовувати в роботі електричну енергію

3.4 Пожежна безпека

Пожежна небезпека – це можливість виникнення та розвитку пожежі в будь-якій речовині, процесі, стані.. Коли людина перебуває в зоні впливу пожежі, то вона може потрапити під дію наступних небезпечних та шкідливих факторів: токсичні продукти згорання, вогонь, підвищена температура середовища, дим, недостатність кисню, руйнування будівельних конструкцій, вибухи, паніка.

Усі працівники повинні вміти користуватись наявними вогнегасниками, іншими первинними засобами пожежогасіння, знати місце їх знаходження.

До первинних засобів пожежогасіння відносяться:

- вогнегасники;
- пожежний інвентар (покривала з негорючого теплоізоляційного полотна, грубововняної тканини або повсті;
- ящики з піском;
- бочки з водою, пожежні відра, совкові лопати) та пожежний інструмент (гаки, ломи, сокири тощо).

Пожежні щити (стенди) встановлюються на території об’єкта з розрахунку один щит (стенд) на площу 5000 м^2. Ящики для піску повинні мати місткість 0,5, 1,0 або 3,0 м3 та бути укомплектованими совковою лопатою.

ВИСНОВКИ

В ході роботи над дипломним проектом було розроблено 2Д гру у жанрі «слешер» на ігровому рушії «Unity». В ході роботи над проектом був проведений аналіз жанру та його відмінних рис. Окрім цього був аналізований ряд вже існуючих проектів в даному жанрі, з ціллю вивчення засобів покращення свого проекту, та можливого переймання та адаптації успішних рішень.

Для реалізації ігрових механік проекту було використано ігровий рушій Unity, а графіку було реалізовано в редакторі піксельної графіки Aseprite. Обидві програми мають весь необхідний функціонал в своїй сфері, для створення проекту. Крім того вони мають дружелюбний до нових розробників інтерфейс і велику кількість освітнього контенту у вільному доступі, що спрощує і прискорює роботу над проектом.

В рамках самого проекту були реалізовані наступні механіки: рух по сцені, який дозволяє гравцеві управляти своїм персонажем; механіка атаки, для персонажа гравця; штучний інтелект ворогів, які переслідуватимуть гравця та, крім того, можуть бути двох різних типів; система генерації ворогів, яка створює певну кількість ворогів кожен раунд. Також для гри була реалізована власна графіка, намальована в стилі «піксель арт»

Даний ігровий проект у майбутньому можна модифікувати, додаючи нові типи ворогів, з унікальними патернами поведінки та/або здібностями. Також не виключено впровадження нових локацій або їх процедурна генерація, що додаватиме грі реіграбельності.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Ерік Фрімен, Елізабет Робсон, Берт Бейтс, Кеті Сієрра, Книга Head First. Патерни проектування, Київ, Фабула, 672 ст.
2. Коноваленко І.В., Програмування мовою C# 6.0, Тернопіль, ТНТУ, 2016 р., 227 ст.
3. Unity Documentation: [Веб-сайт]. URL: https://docs.unity3d.com/Manual/Quickstart2DCreate.html (дата звернення: 28.05.2024).
4. YouTube: [Інтернет-портал]. URL: https://www.youtube.com/playlist?list=PLKGarocXCE1H7pJk6k_CSwS359mtt3MVI (дата звернення: 28.05.2024).
5. Unity Documentation: [Веб-сайт]. URL: https://docs.unity3d.com/ScriptReference/GameObject.html (дата звернення: 28.05.2024).
6. Unity Documentation: [Веб-сайт]. URL: https://docs.unity3d.com/ScriptReference/Collider.OnTriggerEnter.html (дата звернення: 28.05.2024).
7. Unity Documentation: [Веб-сайт]. URL: https://docs.unity3d.com/ScriptReference/Object.Instantiate.html (дата звернення: 28.05.2024).
8. Unity Documentation: [Веб-сайт]. URL: https://docs.unity3d.com/ScriptReference/Random.Range.html (дата звернення: 28.05.2024).

ДОДАТОК А.

Лістинг коду основних модулів гри

Код скрипту «PlayerCtrl»

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class PlayerCtrl : MonoBehaviour
{
    public float movSpeed;
    Rigidbody2D rb;
    private Vector2 axisMovement;

    void Start()
    {
        rb = GetComponent<Rigidbody2D>(); //RigBody for movement
    }

    void Update()
    {
        axisMovement.x = Input.GetAxisRaw("Horizontal");
        axisMovement.y = Input.GetAxisRaw("Vertical");
    }

    private void FixedUpdate()
    {
        Move();
    }

    private void Move() {
        rb.velocity = axisMovement.normalized * movSpeed;
        Check4Flipping();
    }

    void Check4Flipping() {
        bool movingLeft = axisMovement.x < 0;
        bool movingRight = axisMovement.x > 0;
        if (movingLeft) {
            transform.localScale = new Vector3(-4.68f, transform.localScale.y);
        }
        if (movingRight) {
            transform.localScale = new Vector3(4.68f, transform.localScale.y);
        }
    }
}
```

Код скрипту «Health»

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;
public class Health : MonoBehaviour
{
    public int health = 100;
    public int MAX_HEALTH = 100;
    public Spawner Spawner;

    private void Start()
    {
        if (Spawner == null)
        {
            GameObject spawnerObject = GameObject.FindWithTag("Spawner");
            if (spawnerObject != null)
            {
                Spawner = spawnerObject.GetComponent<Spawner>();
            }
        }
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.H)) {
            Heal(10);
        }
    }

    public void SetHealth(int maxHealth, int health) {
        this.MAX_HEALTH = maxHealth;
        this.health = health;
    }

    private IEnumerator VissualIndicator(Color color) {
        GetComponent<SpriteRenderer>().color = color;
        yield return new WaitForSeconds(0.15f);
        GetComponent<SpriteRenderer>().color = Color.white;
    }

    public void Damage(int amount) {
        if (amount > 0)
        {
            this.health -= amount;
            StartCoroutine(VissualIndicator(Color.red));
        }
        else {
            Debug.LogError("Can't have a negative damage");
        }
        if (health <= 0) {
```

```
            Death();
        }
    }

    public void Heal(int amount) {
        if (amount > 0)
        {
 = health + amount > MAX_HEALTH;
            StartCoroutine(VissualIndicator(Color.green));
            if (wouldBeOverMaxHealth)
            {
                health = MAX_HEALTH;
            }
            else {
                this.health += amount;
            }
        }
        else {
            Debug.LogError("Can't have a negative healing");
        }
    }

    private void Death() {
        Debug.Log("I'm dead x_x");
        if (Spawner != null)
        {
            Spawner.enemyCounter--;
        }
        else
        {
            Debug.LogError("Spawner reference is missing in Health script");
        }
        Destroy(gameObject);
    }
}
```

Код скрипту «Enemy»

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Enemy : MonoBehaviour
{
    public int damage = 5;
    public float speed = 1.5f;
    public EnemyData data;
    public GameObject player;

    void Start()
```

```
    {
        player = GameObject.FindGameObjectWithTag("Player");
        SetEnemyValues();
    }

    void Update()
    {
        Swarm();
    }

    private void SetEnemyValues() {
        GetComponent<Health>().SetHealth(data.HP, data.HP);
        damage = data.damage;
        speed = data.speed;
    }

    private void Swarm() {
        transform.position = Vector2.MoveTowards(transform.position, player.transform.position,
speed * Time.deltaTime);
    }

    private void OnTriggerEnter2D(Collider2D collider)
    {
        if (collider.CompareTag("Player")) {
            if (collider.GetComponent<Health>() != null) {
                collider.GetComponent<Health>().Damage(damage);
            }
        }
    }
}
```

Код скрипту «AttackArea»

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class AttackArea : MonoBehaviour
{
    public int damage = 3;

    private void OnTriggerEnter2D(Collider2D collider)
    {
        if (collider.GetComponent<Health>() != null) {
            Health health = collider.GetComponent<Health>();
            health.Damage(damage);
        }
    }
}
```

Код скрипту «PlayerAttack»

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class PlayerAttack : MonoBehaviour
{
    private GameObject attackArea = default;
    private bool attacking = false;
    public float timeToAttack = 0.25f;
    private float timer = 0;

    void Start()
    {
        attackArea = transform.GetChild(0).gameObject;
    }

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) {
            Attack();
        }
        if (attacking) {
            timer += Time.deltaTime;
            if (timer >= timeToAttack) {
                timer = 0;
                attacking = false;
                attackArea.SetActive(attacking);
            }
        }
    }

    private void Attack() {
        attacking = true;
        attackArea.SetActive(attacking);
    }
}
```

Код скрипту «Spawner»

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Random = UnityEngine.Random;
public class Spawner : MonoBehaviour
{
    public GameObject swarmerPrefab, bigSwarmerPrefab;
    private int swarmer = 0, bigSwarmer = 0, roundCounter = 1;
    public int enemyCounter = 0;
    private bool spawning = false;

    private void Start()
```

```
{
    EnemyTypesGenerator(Tickets(roundCounter));
    StartSpawning();
}

private void Update()
{
    if (enemyCounter == 0 && !spawning)
    {
        roundCounter++;
        EnemyTypesGenerator(Tickets(roundCounter));
        StartSpawning();
    }
}

private int Tickets(int rounds) {
    int result = rounds + 10;
    Debug.Log("Сгенерував " + result + " мікетів");
    return result;
}

private void EnemyTypesGenerator(int tickets) {
    while (tickets > 0) {
        int random = Random.Range(1, 3);
        if (random == 1)
        {
            swarmer++;
            tickets--;
            Debug.Log("маленького ворога згенеровано");
        }
        else
        {
            bigSwarmer++;
            tickets--;
            Debug.Log("великого ворога згенеровано");
        }
    }
}

private void Spawn() {
    if (swarmer + bigSwarmer == 0)
    {
        StopSpawning();
        return;
    }
    int random = Random.Range(1, 3);
    if (random == 1 || swarmer != 0)
    {
        //gen swarmer
        Instantiate(swarmerPrefab, new Vector3(this.transform.position.x, this.transform.position.y,
```

```
this.transform.position.z), Quaternion.identity);
            swarmer--;
            enemyCounter++;
            Debug.Log("маленького ворога додано на сцену");
        }
        else
        {
            //gen bigSwarmer
            Instantiate(bigSwarmerPrefab, new Vector3(this.transform.position.x,
this.transform.position.y, this.transform.position.z), Quaternion.identity);
            bigSwarmer--;
            enemyCounter++;
            Debug.Log("великого ворога додано на сцену");
        }
    }

    private void StartSpawning()
    {
        spawning = true;
        InvokeRepeating("Spawn", 2f, 1.5f);
        Debug.Log("спавн почав");
    }

    private void StopSpawning()
    {
        spawning = false;
        CancelInvoke("Spawn");
        Debug.Log("спавн закінчив");
    }
}
```

Код скрипту «EnemyData»

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
[CreateAssetMenu(fileName = "Data", menuName = "ScriptableObjects/Enemy", order = 1)]
public class EnemyData : ScriptableObject
{
    public int HP;
    public int damage;
    public float speed;
}
```

ДОДАТОК Б. Слайди мультимедійної презентації

Розробка 2D-гри у жанрі «слешер» на програмному рушії Unity

Дипломний проект

Студента групи 4РП-07

Лановенко Юрія

Керівник: Джабраілов Д.В.

Особливості ігрового жанру «2D слешер»

- Швидкі та інтенсивні бої.
- Комбінації атак
- Велика кількість різноманітних ворогів

DeadCells

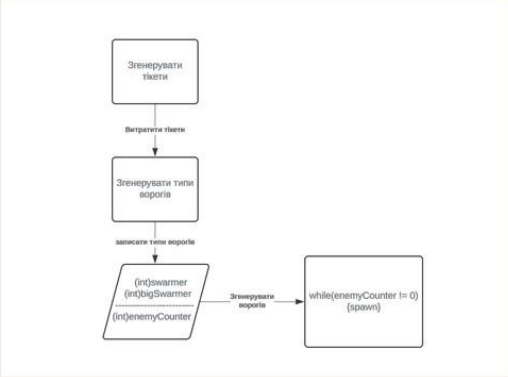
Katana ZERO

Unity: особливості і причини використання

- Hollow Knight: Метроїдванія, Платформер;
- Cuphead: Платформер, Шутер;
- Subnautica: Пригодницький, Виживання;
- Escape from Tarkov: Онлайн-шутер, РПГ;
- Cities: Skyline: Симулятор містобудування;

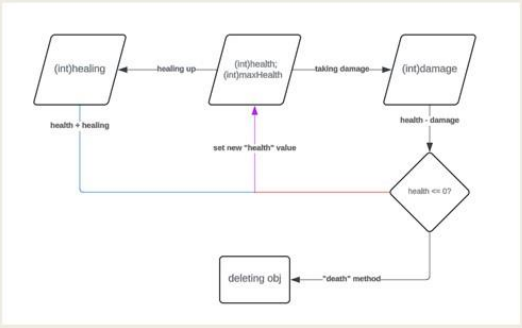
Особливості ігрових механік розроблюємого проекту

Головною особливістю проекту є поступове зростання складності, шляхом збільшення кількості ворогів кожну наступну хвилю



Огляд основних елементів ігрового процесу

Основні елементи ігрового процесу:
- Рух гравця
- Атака гравця
- Переслідування гравця ворогами
- Система генерації ворогів
- Спільна система здоров'я



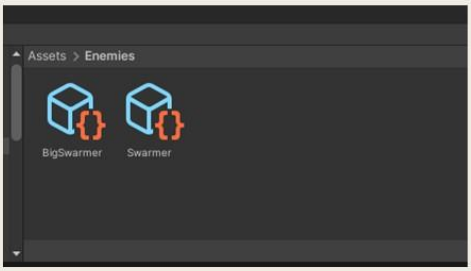
Принцип роботи основних елементів ігрового процесу

Рух:
- Важливість «FixedUpdate»
- Метод «Move»;
- Метод «Check4Flipping».

```
U - Update        -> frame
F - FixedUpdate   -> game/physics step

---U----U----U----U----U----U----U----U----U----U---> time
   F    F    F    F    F    F    F    F    F
```

Система здоров'я:
- Методи «Damage», «Heal»;
- Метод «Death».



Атака гравця:
- Скрипт «AttackArea»;
- Скрипт «PlayerAttack».

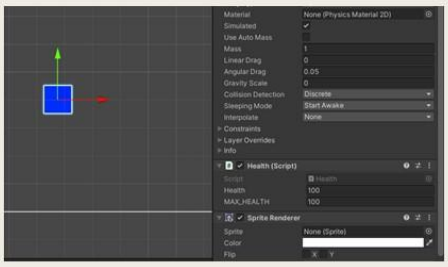
Вороги, їх типи та поведінка:
- Метод «Swarm»;
- Скрипт «EnemyData»;
- Метод «SetEnemyValues».

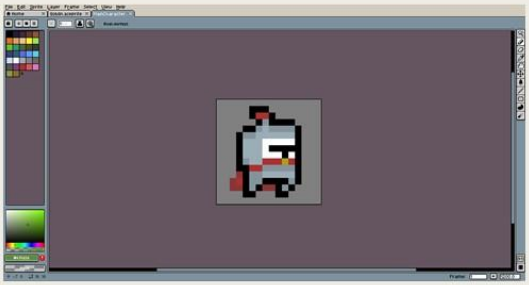
Етапи реалізації основних елементів ігрового процесу

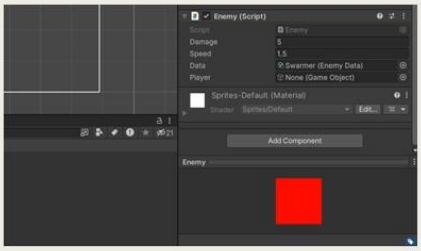
Перший етап:

Розробка ігрової логіки. Створення ігрових об’єктів, розробка і прикріплення до них скриптів, тощо.

Розробка графіки. Малювання спрайтів, їх додання до об’єктів та налаштування.

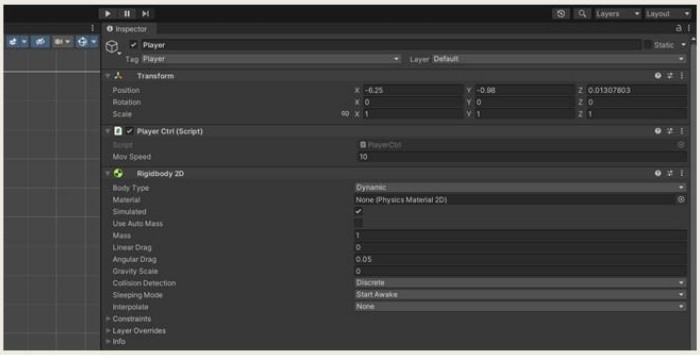






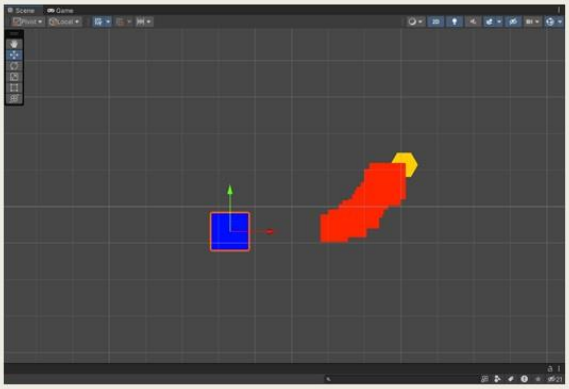
Хід тестування гри

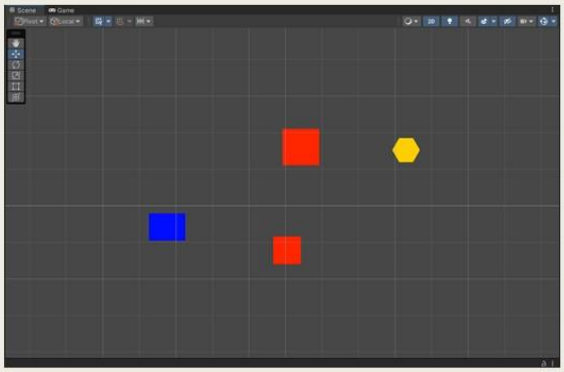
Збір всіх окремих елементів в один робочий проект, для перевірки працездатності і вирішення проблем. Критична важливість даного етапу



Вирішені під час тестування проблеми:

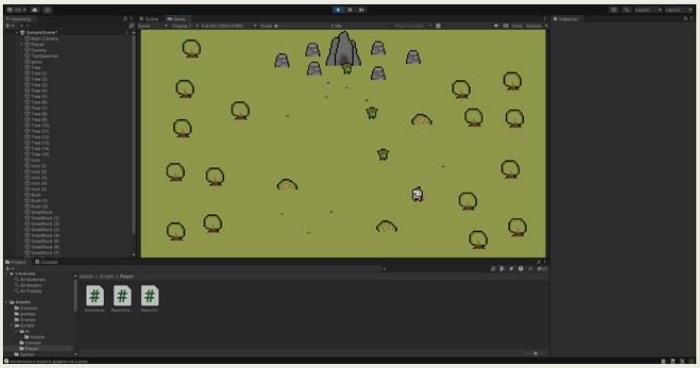
- Розтягування моделі гравця;
- Неконтрольованння генерація ворогів;
- Генерація ворогів не там де потрібно.





Скріншот гри:

Зображеня тестування ігрового процесу в умовах ігрового рушія Unity



ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Лановенко Юрія Олександровича

(прізвище, ім'я та по батькові)

Спеціальність: 121 "Інженерія програмного забезпечення"

Освітня програма: «Розробка програмного забезпечення»

Тема дипломного проекту: Розробка 2D-гри у жанрі слешер 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 77 сторінок. У пояснювальній записці виконано опис предметної області, способи реалізації 2D-гри в жанрі слешер. Проведено проектування та реалізацію всіх важливих елементів 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 Лановенко Ю.О. 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 Лановенко Ю.О. 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 Лановенко Ю.О. мав змогу самостійно приймати рішення з реалізації необхідних елементів 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

Лановенко Юрія Олександровича

(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 слешер на програмному рушії Unity*

Обсяг розрахунково-пояснювальної записки 77 сторінок

Обсяг графічної (презентаційної) частини 10 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. Детально 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. Не реалізовано деякі етапи інтерактивної взаємодії з предметами оточення під час ігрового процесу;
2. На ігрових рівнях реалізовано не дуже багато варіацій ворогів.
3. Друкована робота містить деякі помилки оформлення.

Оцінка розрахункової частини Відмінно

Оцінка графічної частини Добре

Загальна оцінка Відмінно

Прізвище, ім'я, по батькові рецензента к.т.н. Кірєєв Ігор Анатолійович

Місце роботи і посада рецензента Державний університет інтелектуальних технологій і зв'язку, доцент каф. інформаційної безпеки та передачі даних

Підпис:

ПІДПИС ПОСВІДЧУЮ
НАЧАЛЬНИК ВІДДІЛУ
КАДРІВ ДУІТЗ

ВІДДІЛ КАДРІВ

«14» 06 2024 р.

UNICHECK
by Turnitin

Ім'я користувача:
Катерина Григоріївна Краснокутська

ID перевірки:
1016350946

Дата перевірки:
12.06.2024 09:21:30 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
12.06.2024 09:22:17 EEST

ID користувача:
100011688

Назва документа: 4РП-07_Лановенко

Кількість сторінок: 50 Кількість слів: 8114 Кількість символів: 57081 Розмір файлу: 2.59 MB ID файлу: 1016154624

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

7.19%
Схожість

Найбільша схожість: 4.62% з Інтернет-джерелом (https://card-file.ontu.edu.ua/server/api/core/bitstreams/6c95086b-bff...

7.19% Джерела з Інтернету | 174 ..... Сторінка 52

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 | 1

Підозріле форматування | 10 сторінок

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

Ми, що нижче підписалися,

Лановенко Юрій Олександрович,
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 слешер на програмному рушії Unity» (автор роботи – Лановенко Ю.О., 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 _____ / Лановенко Ю.О./

Керівник _____ / Джабраілов Д.В./

«10» 06 20 24 р.